Magisterarbeit

zum Thema

Plausibility of Databases and the Relation to Imputation Methods

ausgeführt am
Institut für Statistik und Wahrscheinlichkeitstheorie
der Technischen Universität Wien

unter der Anleitung von
A.o. Prof. Dipl.-Ing. Dr.techn. Peter Filzmoser

durch
Heinrich Fritz
Matr.Nr.: 0325693
Landstraßer Hauptstraße 104/4
1030 Wien

Wien, am 19. April 2007

Preface

Motivated by the omnipresent issue of missing values in databases, the development of new approaches for estimating missing data is always subject of research, especially in departments with focus on applied statistics as the "Department of Statistics and Probability Theory" at the TU Wien, Austria.

The fundaments of this thesis was the aim of developing novel robust imputation methods in combination with principal component analysis. Since there is hardly any literature on this topic, a paper of Sven Serneels et al. [7] has been considered as initial point for this development. Originally the proposal of Serneels was only intended as inspiration for further development, however several aspects have been identified which lead to an improved version of the algorithm proposed by Serneels, with respect to missing data estimation. Aside the mathematical prerequisites of imputation based on principal component analysis, the development of this improved method with all its issues has been summarized in this thesis.

However not only the theoretical background has been treated in this context, but also an implementation of the mentioned methods, which has been optimized regarding to runtime behaviour and memory consumption. Since robust methods can usually be characterized as computational intensive, efficient implementation is one of the key aspects in this context in order to be able of applying these methods on large amounts of data. As the environment R is methodically very powerful and mature, it cannot be considered as economical in terms of resource consumption. To avoid these drawbacks critical components are usually implemented in C++ or Fortran. So the mathematical development of new methods in combination with the challenging implementation with respect to limited capacities, were the reasons for my decision to write this thesis as conclusion of my studies in informatics.

Contents

Chapter 1

Introduction

Nowadays almost every process which deals with any kind of information, is based upon a database containing the data this information is made of. No matter what kind of automated task is considered, in the background a database keeps and manages all the data which is necessary for keeping things running. While issues like consistency, reliability, stability, etc. of databases have been treated amply, there are only few approaches which check the contents of databases for plausibility. Not only the basics of keeping such a system alive and intact are essential, but also the quality of contents affects the efficiency of the tasks and procedures which are running in such environments.

This thesis will mainly focus on plausibility of the contents of databases. On the one hand, methods for estimating the plausibility of single entries with respect to the whole data structure will be presented, and on the other hand, methods for estimating missing data will be discussed and improved. Since missing values are one of the main issues of data aquirement, efficient algorithms for estimating these values are conductive for improving the quality of a database, regarding to its contents. Since the field of such *imputation* methods as described by Little and Rubin (2002) is very wide, this thesis will mainly focus on single imputation methods, dealing with continuous data. The development of methods for categorical data and multiple imputation based on the methods discussed here, is already in progress, but would go far beyond the scope of this thesis.

The fundament for the estimation of missing data is in this context a paper, written by Walczak et al. (2001), and Serneels et al. (2007) who propose a method for principal component analysis on data containing missing values. After discussing the theoretical background of the mentioned issues, the approach of Walczak et al. (2001) will be adopted and a rather new imputation method will be presented.

An aspect which will be omnipresent is robustness. Since single entries from a database, which do not fit the general structure, shall not distort any estimations and assumptions based on these data, the considered methods and algorithms are designed to minimize the influence of such outlying entries, in order to obtain results representing the structure which stands behind the majority of observations. Croux et al. (2007) as well as Rousseeuw et al. (1999) have developed several methods, which will be very useful in this context.

Chapter 2

Plausibility of Databases

2.1 Databases in Statistical Context

Since this thesis deals with algorithms, applied on the contents of databases, first of all, the way databases are treated in this context should be specified exactly. In contrary to a database oriented approach, tables will not be considered as entities with relations to each other but such multivariate data structures are represented by matrices. If not specified explicitly these matrices are of size n x p which means, the underlying entity had n entries and p attributes. In our context the attributes are considered as dimensions. This means, that each entry or observation from our entity is located at an exact defined position in a p dimensional space.

Further the content has to be adapted in order to fit the prerequisites of mathematical algorithms. In general only numerical values can be processed by these routines. Data which is usually represented by strings or other not exact defined data types has to be converted to numerical values. This applies usually to categorical data (e.g. gender, colour of eyes, ...) which has to be coded in some way (e.g. *male* = 1, *female* = 2 or *blue* = 1, *green* = 2, *brown* = 3, ...). Of course the original meaning of the coded values is kept in order to interpret the results of miscellaneous methods appropriately, but it should be mentioned here, that most of the algorithms can only deal with numerical data and therefore the data format has to be adjusted. Considering R (www.r-project.org) as a tool which has been designed for statistical research and calculation, this functionality has already been implemented and the user does not need to care about the representation of categorical data any longer. Nevertheless in some cases it may be very helpful to be aware of these prerequisites and the way information is coded internally. For instance, if the considered data contains local information which is coded by addresses, a conversion into geographic coordinates which represent exactly the same information may be necessary for the mathematical algorithm, in order to be able to interpret the local relations between single observations correctly.

In general variables can be categorized into three basic types:

- **Nominal variables**
  Simple names are assigned to objects. These names may be coded by numbers, however ranking these values would not make sense at all. (e.g. colors, countries, gender, ...)

- **Ordinal variables**
  Similar to nominal variables, but the values can be ordered and compared to each other (e.g. grades at school). Simple methods like median calculation are possible in this context.

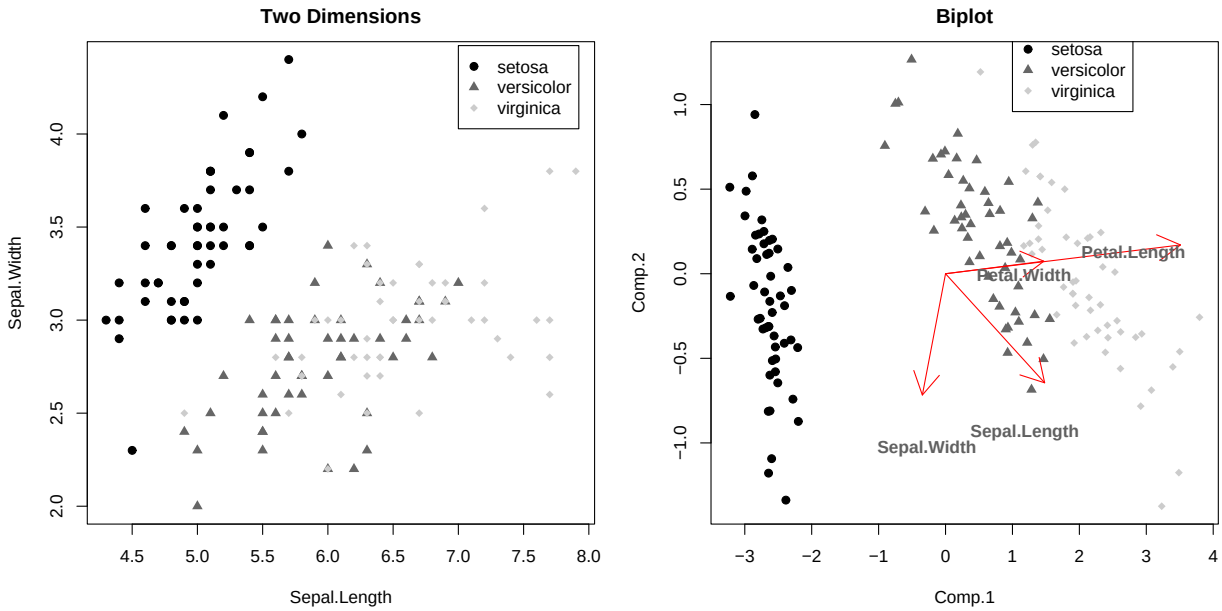
- **Continuous variables**
  In contrary to the other types of variables, which are also called categorical types, this type may contain all values $\in \mathbf{R}$ or a subset of $\mathbf{R}$. Further these values may be compared to each other, as well as added, multiplied and divided by each other (e.g. length, mass, speed, ...).

2.2 Basic Diagnostic Plots by Data Projection

A very common method for getting a first impression of a dataset is the so called biplot as proposed by Gabriel (1971), shown in Figure 2.1 (b). The plot shows data of different types of iris flowers. The underlying data set contains four variables, named *Sepal.Width*, *Sepal.Length*, *Petal.Width* and *Petal.Length*, which are characteristic for these flowers. Further a categorical variable indicating the species of iris (Iris *Setosa*, *Versicolor* and *Virginica*) is available. As ordinate and abscissa, the first two *principal components* are chosen, which are two orthogonal directions in our 4-dimensional space. These directions are chosen in a way, that a scale measure of the data projected onto these directions is maximized. Such directions can be determined by *principal component analysis* or *PCA* (Hotelling 1933). In other words, the biplot shows the data from a direction which is very informative and shows much of the structure of the data. Compared to this approach we also could plot two variables against each other as shown in Figure 2.1 (a). In this plot the structure of the data is not as clear as on the right side. Without any knowledge about the type of the observations, we could easily identify the *setosa*-type at the very left edge of the biplot, which is much more difficult in (a). Further, two other species may be distinguishable without additional information in the biplot. When considering only plot (a), one would never be able to identify the three groups of iris flowers correctly without any additional information.

The system of coordinates in the biplot intimates, that it is usually influenced by each single variable, whereas a two dimensional plot obviously depends only on two variables. This system of coordinates indicates the influence of the single variables on the plot itself, and gives additional information concerning the correlation of the variables. If two axes point into a similar direction, one can assume that these variables are highly positive correlated. Axes pointing into opposite directions indicate a negative correlation, and a pair of approximately orthogonal axes indicates a very low or no correlation.

Of course one could plot each pair of variables separately, but especially when processing



(a) (b)

Figure 2.1: Fisher's iris data set. Two dimensional plot (a) vs. biplot (b)

high dimensional data, a simple biplot gives much more information on the structure than a pairwise comparison of the variables. The concept of biplots can be extended a little by adding plots of the first versus the third or the second versus the third principal component. This may give a more detailed and comprehensive insight in the structure.

A first, very brief check of our data using the biplot does not give any information on *strange* observations which do not follow the general structure of the data. However if there was an observation at the very right in the biplot which actually belongs to the *setosa* group, this would be an indicator for some irregularities or contamination in our data. In this case one should have a closer look at the concerned observation and, if possible, figure out where this irregularity comes from.

2.3 Principal Component Analysis

Talking about principal components, we should first exactly specify the mathematical model which stands behind, and discuss various possibilities of calculating these components - the different kinds of principal component analysis. As already mentioned PCA tries to find directions in the data space which maximize a scale measure of the data projected onto these directions. Usually PCA is carried out with centered data matrices which requires a center estimation in advance. From now on the estimated center of a data matrix $\boldsymbol{X}$ with n rows and p columns will be denoted as $\boldsymbol{T}$ and the estimated covariance as $\boldsymbol{C}$, regardless of the estimation method. Appropriate methods for centering multivariate data will be discussed later on (Section 2.4.1).

Further scaling the data in a preceding step may also be useful due to different ranges of the variables. Considering geometric characteristics of objects, it appears obvious that the variables should represent the same units (e.g. *mm*). If variables which are coded differently (let's say in *mm* and *m*) are mixed, the *mm* measures would dominate the model due to the high values in comparison to *m*-measures. To avoid this problem, scaling in advance is an appropriate approach. Classical PCA methods require an estimation of the covariance structure in advance, which explains pairwise relations between the variables. The covariance matrix of a data set $\boldsymbol{X}$ is classically estimated by the sample covariance matrix

$$\boldsymbol{S} = \frac{1}{n-1} \sum_{i=1}^{n} (\boldsymbol{x}_i - \bar{\boldsymbol{x}}) (\boldsymbol{x}_i - \bar{\boldsymbol{x}})^\top \tag{2.1}$$

where $\boldsymbol{x}_i$ are the p-dimensional observations and $\bar{\boldsymbol{x}} = \frac{1}{n} \sum_{i=1}^{n} \boldsymbol{x}_i$ is the arithmetic mean vector. However, both location and covariance can be estimated in a different way, denoted by $\boldsymbol{T}$ and $\boldsymbol{C}$, respectively. From now on we will consider $\boldsymbol{X}$ as already centered and scaled data matrix, which makes the formal explanation of PCA a little bit easier. PCA is usually based on covariance decomposition:

$$\boldsymbol{C} = \boldsymbol{\Gamma} \boldsymbol{\Lambda} \boldsymbol{\Gamma}^\top \tag{2.2}$$

The covariance matrix is decomposed into its eigenvectors, columnwise represented by $\boldsymbol{\Gamma}$ and its eigenvalues, represented by the diagonal matrix $\boldsymbol{\Lambda}$ as its main diagonal. By calculating the eigenvalues the largest extent of the ellipsoid which is given by the covariance matrix is being searched. This extent is the first eigenvalue, and the corresponding direction gives the first eigenvector. The second largest extent of this ellipsoid, orthogonal to the first direction which has been determined, gives the second eigenvalue and so on. As our data matrix $\boldsymbol{X}$ was of dimension $n \times p$, ($\boldsymbol{\Gamma}$) containing p eigenvectors of length p is of dimension $p \times p$, as well as $\boldsymbol{\Lambda}$ with its p eigenvalues, arranged on its main diagonal. Each eigenvector contained in $\boldsymbol{\Gamma}$ gives the direction of one principal component and the corresponding eigenvalue of $\boldsymbol{\Lambda}$ defines its length. As the eigenvalues are in descending order, the first eigenvector and eigenvalue represent the largest principal component, the last eigenvector and corresponding eigenvalue represent the smallest principal component.

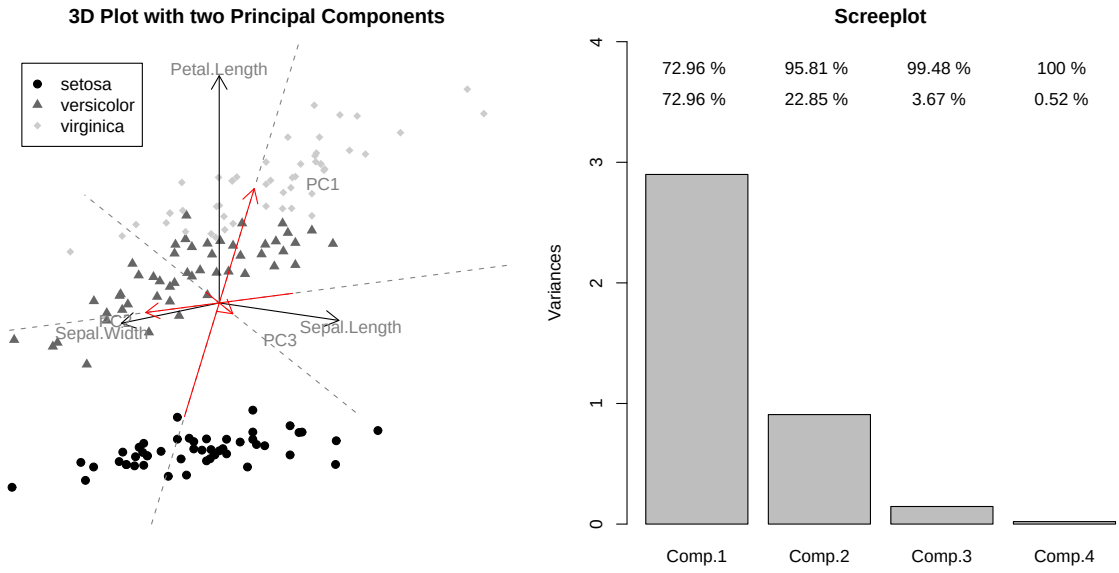
As seen in Figure 2.2 (b), the first principal component explains 73% of the variance contained in the data. Since all principal components are orthogonal to each other, they can simply be considered as a rotated system of coordinates. $\boldsymbol{\Gamma}$ which is orthogonal, is the corresponding rotation matrix which performs this transformation of the system of coordinates:

$$\boldsymbol{Z} = \boldsymbol{X} \boldsymbol{\Gamma} \tag{2.3}$$

whereas $\boldsymbol{Z}$ is called *scores* and represents the rotated data matrix. These scores contain exactly the same information as $\boldsymbol{X}$, only represented in a different way. The rotation matrix is usually called *loadings*. Since it is orthogonal, rotating the scores back to the original data is very simple:

$$\boldsymbol{X} = \boldsymbol{Z} \boldsymbol{\Gamma}^\top \tag{2.4}$$

The advantage of expressing the whole data structure by these scores, is that compared to the original structure, the information is distributed differently among the columns. As

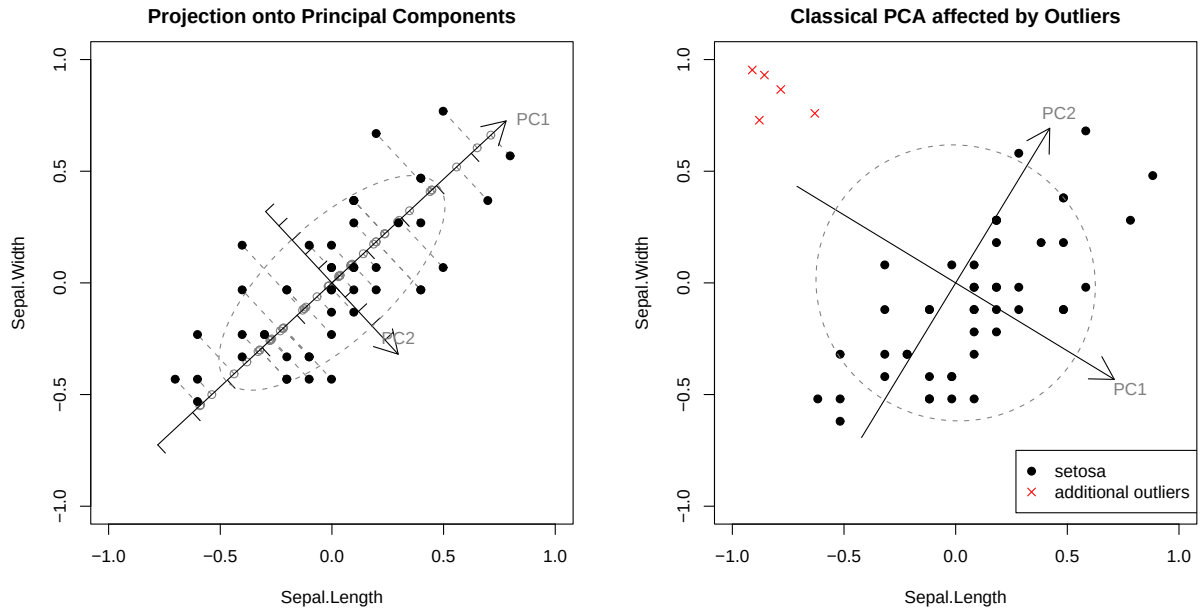


(a) (b)

Figure 2.2: Fisher's iris data set. (a): Plot of the first 3 variables of the dataset including the fist two principal components and a screeplot of the same dataset (b).

described in Figure 2.2 (b), the third and fourth principal component only describe 2.5% of the variance, which means, that 95% of the information of the original data matrix is contained in the first two columns of the scores. So by dropping the third and fourth column of the scores, we make very little mistake and can reduce the amount of data needed to represent 95% of the information by half. Of course data compression is not the main purpose of PCA but in certain cases this may be helpful.

By simply dropping the last few columns of $\boldsymbol{Z}$, the data is being projected onto a hyperplane spanned by the remaining principal components. The number of principal components which contain relevant information is in this context denominated as $k <= p$. Figure 2.3 (a), shows an example of this projection in two dimensional space, where all observations are projected onto the first principal component ($p = 2, k = 1$). Again both systems of coordinates are displayed: Outside the original system which has been centered, and the inner axes representing the rotated system. The projection onto the principapl components is done by setting the scores of the second principal component to zero or by dropping the according columns of $\boldsymbol{Z}$ which is equal. Compared to the projection done in the original system of coordinates this is very simple. Afterwards the projected data has to be rotated back which is done by Equation (2.4). If we had set column $k+1, \ldots, p$ of $\boldsymbol{Z}$ to zero this fits perfectly, however if we have dropped these columns completely, we have to drop them in $\boldsymbol{\Gamma}$ too, in order to be able to use (2.4) which simply results from the prerequisites of matrix multiplication. Mathematically this is the same, but becomes important during implementation of this back-transformation.



(a) (b)

Figure 2.3: Fisher's iris data set. *Sepal.length* vs. *sepal.width*, only showing observations of type *setosa*

2.3.1 Number of Relevant Components

A question which arises in this context is the determination of k - the number of relevant components in the structure. The aim is to represent the data with as few components as possible and a minimum of loss of information. These are two quite contradictory ideas and require a sort of trade-off between the number of components and loss of information. In the introducing example the decision was rather easy as dropping 50% of the components only resulted in 5% loss of information which is quite unusual. Normally it is very hard to make a clear decision in this context. In literature several rules are proposed which are in fact only rules of thumb, based on experience and not on statistical methods or facts. A rule for example would be to chose as many components so that 90% of the variance is explained. Following another rule one has to drop the components which explain less variance than the medium variance explained by all components. There are numerous approaches of this kind which all have been developed in a specific context and may have been appropriate but it is not possible to formulate a general rule which solves this problem.

2.4 Robustification of PCA

As shown in Figure 2.3, (b) PCA as it has been explained up to this point is very sensitive to outliers. The plot shows some artificially added outliers which contaminate the dataset and distort the original principal components in a way, that they change their directions

completely. Further one can see, as intimated by the grey dashed ellipsis, that the covariance structure has changed tremendously. The system as shown in (b) is very instable, because no matter which direction is considered, the variance of observations projected onto this direction is quite constant - the corresponding ellipse is nearly a circle. In such a case a slight change of the data may result in a totally new constellation of principal components which is of course not desired. Robustification of this method tries to reduce the influence of some individual observations in order to gain a result which is only influenced by the majority of the data.

There are two main reasons which are responsible for this sensitive behaviour: One the one hand the classical mean, which is here used as center estimation, and moreover the classical covariance estimation is very sensitive to single outliers. Since deviations from the center are squared and so affect the classical covariance estimation tremendous it is obvious, that this approach cannot be appropriate in such a context. Solutions to this issue are still object of research, and permanently new approaches are developed for robustifying existing methods.

A first approach of robustifying PCA is to choose robust location estimates for centering, and robust covariance estimates for determining the principal components. The advantage of this approach is, that the development of these estimators is completely independent of PCA which only requires centered data and information on the covariance structure regardless of the estimation method. In context with robust methods a term which occurs quite often is the so called *breakdown point*, specifying the amount of outliers which may be contained by the dataset without resulting in non-sense estimation.

The following section gives a short overview of common robust location and scale, as well as covariance estimators:

2.4.1 Robust Location Estimation

As alternative to the classical mean

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i \tag{2.5}$$

which is affected easily by outliers, the following location estimates are usual:

- The **median** of a sample $x = (x_1, x_2, \ldots, x_n)$ has a breakdown point of 50% and is defined as

$$\tilde{x} = \begin{cases} x_{(n+1)/2}, & n \text{ odd} \\ \frac{1}{2}(x_{n/2} + x_{n/2+1}), & n \text{ even} \end{cases} \tag{2.6}$$

  However this estimate is not optimal in a multivariate environment because the location estimation is done for each column separately and not with respect to the whole multivariate structure.

- The L_1**-median**, also known as spatial median estimates the center of the structure as it returns the point with the minimum distance to all observations:

$$\sum_{i=1}^{n} \|\boldsymbol{x}_i - \boldsymbol{T}\| \rightarrow \mathrm{m}in \tag{2.7}$$

  As this concept is derived from the classical median, the L_1-median has a 50% breakdown point as well.

2.4.2 Robust Scale Estimation

Scale estimation apart from covariance estimation has not been considered yet but will be very important in the subsequent chapters dealing with alternative approaches of PCA.

Instead of the classical standard deviation

$$\mathrm{SD}(x) = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2} \tag{2.8}$$

one of the following robust scale estimates may be used:

- **Median Absolute Deviation** with breakdown point of 50%, also known as MAD. The MAD of a sample x is defined as

$$MAD\,(x) = \underset{i}{\mathrm{med}}(|x_i - \tilde{x}|) \tag{2.9}$$

  If the underlying distribution can be assumed as normal distribution, the MAD has to be adjusted in order to be comparable with SD estimates:

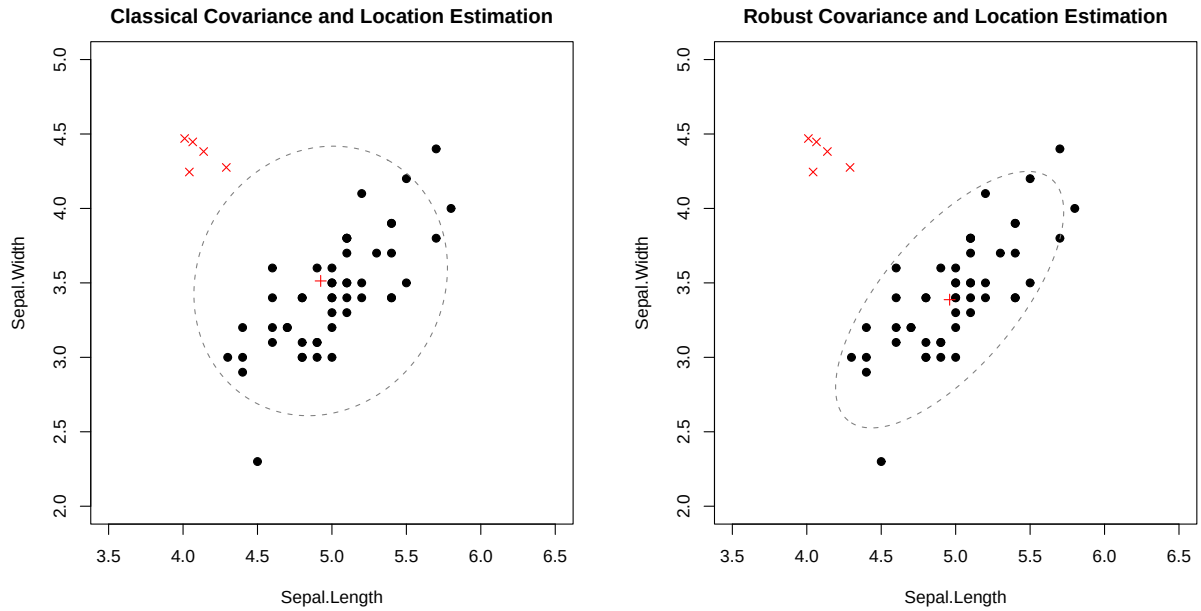
$$S_{MAD} = \frac{MAD}{0.675} \tag{2.10}$$

- Q_N as proposed by Rousseeuw and Croux (1993) is an even more efficient alternative and defined as

$$Q_N(x) = d\left\{|x_i - x_j|; i < j\right\}_{(k)} \tag{2.11}$$

$$S_{QN} = 2.2219 Q_N \tag{2.12}$$

  whereas 2.2219 is again an adjusting factor for compatibility with SD estimates under normal distribution and $k = \binom{h}{2} \approx \binom{n}{2}/4$ and $h = \left[\frac{n}{2}\right]$.



(a) (b)

Figure 2.4: Fisher's iris data set. *Sepal.length* vs. *sepal.width*, only showing observations of type *setosa*. Covariance and location estimation.

2.4.3 Robust Covariance Estimation

A common approach of covariance estimation of multivariate normally distributed data is the *Minimum Covariance Determinant* (*MCD*) estimator, described in Rousseeuw and Leroy (1987). The idea behind is to find an ellipsoid with minimal determinant which covers at least h observations whereas $\lfloor \frac{n+p+1}{2} \rfloor \leq h \leq n$. The choice of h determines the efficiency which increases with higher values of h, as well as the breakdown point which is $\frac{n-h}{n}$ and decreases with increasing h. Again here has to be made a compromise between high breakdown point and efficiency.

Since considering each possible set of h out of n observations is computational too intensive for processing larger amounts of data, Rousseeuw and van Driessen (1999) have developed "*A fast algorithm for the minimum covariance determinant estimator*" which can handle sample sizes up to $n = 50.000$, which was, apart from its efficiency and robustness, decisive for this method to become popular.

The procedure proposed by Rousseeuw and van Driessen (1999) mainly consists of two steps:

- h observations are chosen randomly which define the index set I_1. The classical center $\bar{\boldsymbol{x}}_1$ and covariance structure $\boldsymbol{S}_1$ is estimated only considering these h values. Further the Mahalanobis distance of each observation is calculated, which is defined as:

$$\mathrm{MD}(\boldsymbol{x}_i) = \sqrt{(\boldsymbol{x}_i - \boldsymbol{T})^\top \boldsymbol{C}^{-1} (\boldsymbol{x}_i - \boldsymbol{T})} \qquad , i = 1, \ldots, n \tag{2.13}$$

wheras $\boldsymbol{C}$ is replaced by $\boldsymbol{S}_1$, and $\boldsymbol{T}$ by $\bar{\boldsymbol{x}}_{\boldsymbol{1}}$ in this situation.

- All observations are sorted according to their Mahalanobis distance, and a new index set I_2 is chosen which contains the h observations, with the smallest distance estimation. Again the classical covariance $\boldsymbol{S}_2$ and location $\bar{\boldsymbol{x}}_2$ are estimated.

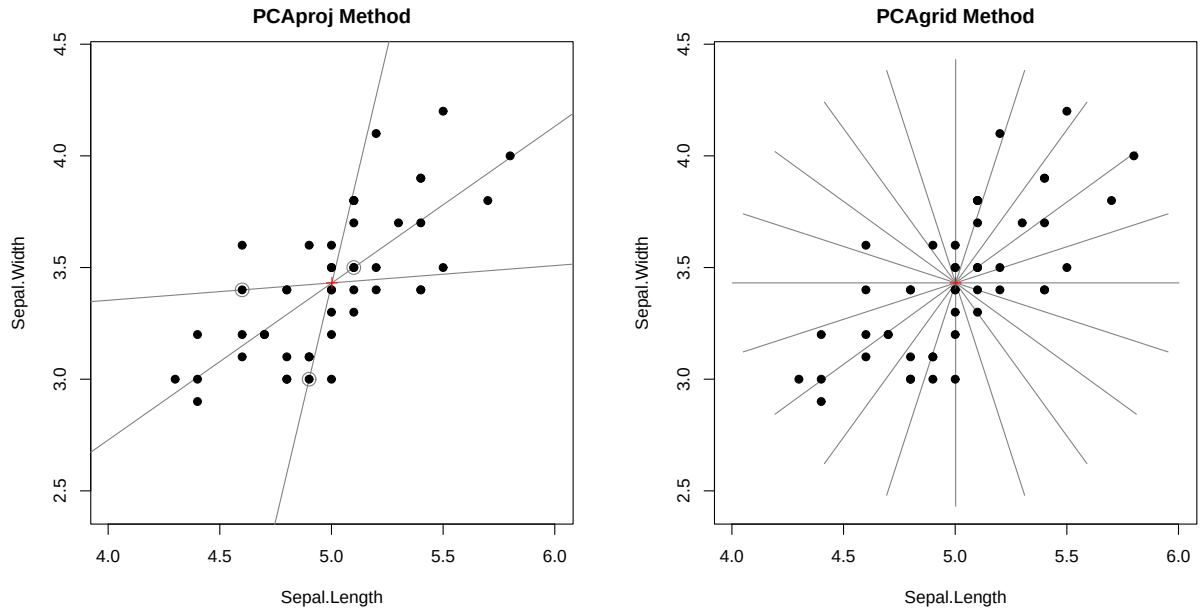
Since $\det(S_2) \leq \det(S_1)$, repeatedly applying these steps continuously reduces the determinant until convergence. The obtained location and covariance estimate can then be considered as representing the robust structure of the whole data set. In order to avoid local optima, the whole algorithm is applied repeatedly with different initial index sets $\boldsymbol{I}_1$, and the covariance estimation with smallest determinant is then chosen as final result.

Figure 2.4 shows the difference between the classical (a) and the robust (b) covariance and location estimation. The robust estimation has been done with the MCD algorithm as described above. As the covariance structure on the left is strongly attracted by few outliers, the structure on the right is stable, and represents the majority of the observations, which are in this case the *real* observations, quite well.

2.5 PCA by Projection Pursuit

Robustification of PCA as described above is quite useful in certain cases, but there are circumstances which make the underlying estimators fail. These drawbacks were the motivation of the development of even more powerful PCA methods as described by Croux et al. (2007). The idea was to avoid the covariance decomposition which is fundamental for classical PCA and to determine the single components by projection pursuit. The advantage of this approach is, that it is especially useful for flat data matrices ($n \leq 2p$) and for very large, high dimensional data sets. The first issue concerns covariance estimates which always need a minimum number of observations, usually ($n \geq 2p$) is required, and since covariance estimation is not necessary in this context, this restriction can be ignored. The second advantage is the iterative manner the components are determined. Classical PCA, regardless of the underlying estimation methods, always calculates all components at once. The projection pursuit approach finishes with the first component before processing the second component and so on. If only a certain amount of components is needed, the algorithm can be stopped, which is a great saving of time, especially when computing big data matrices.

The basic idea of projection pursuit is to project the data onto a hyperplane and watch the change of the structure of these projected data while the hyperplane is rotating. In our case we want to maximize a scale measure of our projected data, which gives us then the desired principal components. So what has to be done, is to choose a direction in our p-dimensional space and try to rotate it in a way that a chosen scale measure is being maximized. Croux and Ruiz-Gazen (2005) suggest the MAD or Q_N estimator for this purpose. The problem which arises at this point, is the incredible amount of possible directions which have to be checked, so there must be rules decreasing this amount to a reasonable set of directions,



(a) (b)

Figure 2.5: Fisher's iris data set - only showing observations of type *setosa*. Candidate directions for *PCAproj* (a) and *PCAgrid* method (b).

which can be considererd as candidate set for the principal components.

Since all principal components point through the center of the data, the first rule is obvious which eliminates all directions that do not fit this prerequisite. This reduces the amount of candidate directions enormous but their number is still infinite. The R package *pcaPP* implements this method and supports two types of direction selection:

- **PCAproj**
  The first method, illustrated in Figure 2.5 (a), as suggested by Croux and Ruiz-Gazen (2005) is called *PCAproj* and basically focuses on directions which directly point through observations. So as first approximation only directions are checked which point through the center and single observations. When a proper candidate for the first principal component is found, the search is being continued in the orthogonal subspace to the found component until no dimensions are left or the desired number of principal components is reached. As mentioned by Croux et al. (2007) this algorithm has serious drawbacks concerning the eigenvalues, which may drop to zero, especially when computing very flat matrices and using robust scale estimates. Considering the orthogonal subspace of a principal component, which is defined by a specific observation, this observation is exactly in the center of the new subspace. Considering a dataset with few observations and many dimensions, the estimation of the k-th principal component is strongly disturbed, since $k-1$ observation are exactly located in the center of the considered subspace, which leads to an effect, called *implosion of scale*.

A known solution to this problem is an updating step, which adjusts the principal components afterwards, in order to increase the quality of the result. The components are rotated and shifted slightly in a way, that observations which have been used as candidate directions are not located exactly on the resulting components, so the problem of implosion of scale can be reduced.

Another possibility, which has been implemented in the mentioned package, is to add additional observations which can be chosen randomly or generated as linear combinations of the existing observations. These additional observations are obviously not considered, when calculating the scale estimate which is being maximized. However, they help to stabilize the system, since the chance, that a principal component exactly points through an observation and thus the effect of implosion of scale can be reduced.

- **PCAgrid**
  The grid algorithm, as proposed by Croux et al. (2007), avoids these problems by following a completely different approach: Variables are ordered decreasing by their scale in advance, and the optimization is only done in a two dimensional subspace spanned by the first two variables instead of the whole p-dimensional space which is computational less intensive. As shown in Figure 2.5, the plane is separated into a fixed number of sectors and the direction with the highest scale estimate is chosen. The procedure is repeated for the first and third variable, the first and the fourth, ..., until the first principal component is found. In the subspace orthogonal to the first principal component the second principal component is evaluated in the same way. Since this approach is not oriented on single observations which are considered as candidate directions, the chance that a principal component exactly points through an observation is very small, thus the problem of implosion of scale negligible.

  Concluding it should be mentioned, that the grid algorithm delivers practically the same result as classical PCA, when classical location and spread estimates are used for determining candidate directions.

As already mentioned, the computation of principal components using the projection pursuit approach is not based on any preceding estimation of a covariance structure. Robust covariance estimation (e.g. by the MCD) can only be done when $n \geq p$ and the considered robust methods do even need a larger set of observations where approximately $n \geq 2p$ is required. Still many common multivariate methods depend on preceding covariance estimation which makes them useless in the considered case of rather flat data matrices.

At this point PCA by projection pursuit can help, as described in Equation (2.2). A covariance estimation can be done, by simply using the calculated loadings and lengths of the principal components. Thus when a PCA method has been applied on a dataset, which does not need a covariance estimation in advance, PCA can be *reversed*, in order to obtain an estimation for the covariance structure. Even when $n < p$ a covariance estimation can be done, with the obvious restriction, that the resulting covariance matrix is only of rank $n-1$.

2.6 Outlier Detection

After discussing the principals of multivariate statistics, and the robust analysis of data sets, these methods shall now be used to identify observations, which do not belong to the main structure represented by the data. One of the central elements in diagnostic is the covariance structure, as the *good* observations are located corresponding to it. As already shown in 2.4.3, outlying observations can be identified by considering the Mahalanobis distance. This measure represents the distance of an observation from the estimated center of a dataset with respect to the covariance structure. Obviously we will consider a robust covariance estimate, since calculating Mahalanobis distances based on a classical covariance estimation, usually leads to masking. Masking is an effect which appears, when the covariance structure is influenced by strongly outlying observations in a way, that other outlying observations appear as they belong to the general data structure.

In Figure 2.6 a data set is shown, which contains body- and brain weights of 28 animals. Plot (a) and (c) show the logarithm of theses values which are highly correlated. In plot (a), the robust covariance structure is represented by a dashed ellipse. All points on this ellipse have a Mahalanobis distance of $\sqrt{\chi^2_{2,0.95}} = 2.45$, and since the squared Mahalanobis distances of multivariate normally distributed observations are χ^2_p- distributed, all observations outside a certain quantile (e.g. 0.95) can be considered as outliers. In this example five observations point out, three of them with lower brain mass than the average, and two observations with higher brain mass compared to their body weight. Plot (b) shows the Mahalanobis distances of each observation, where again these five outliers can be seen. The group containing three outliers can here be identified as dinosaurs. Since it is very likely, that observations which represent dinosaurs do behave different in this context, it seems plausible, that they point out in such a diagnostic plot. Two other outliers which have been detected have Mahalanobis distances barely larger than the considered threshold and represent Human and Rhesus monkeys. When lowering the distance threshold, the next observation which points out stands for chimpanzees, so another group can be identified, which represents primates. Since primates are in general more intelligent than the majority of animals, this is also coherent.

By considering robust Mahalanobis distances we were able to identify two groups which consist of outlying observations, and by having a second look, we were even able to ascertain the reason for this behaviour. When considering the covariance structure, estimated in the classical way, as shown in plot (c), we can also identify two observations as outliers, which again are dinosaurs. However, since we know that our dataset contains five observations which do obviously not belong to its main structure, we have to assume, that classical covariance estimation in this context is insufficient. Some observations, which here represent dinosaurs distort the covariance structure in a way, that observations which are in fact outliers do appear as regular observations, which is called masking.

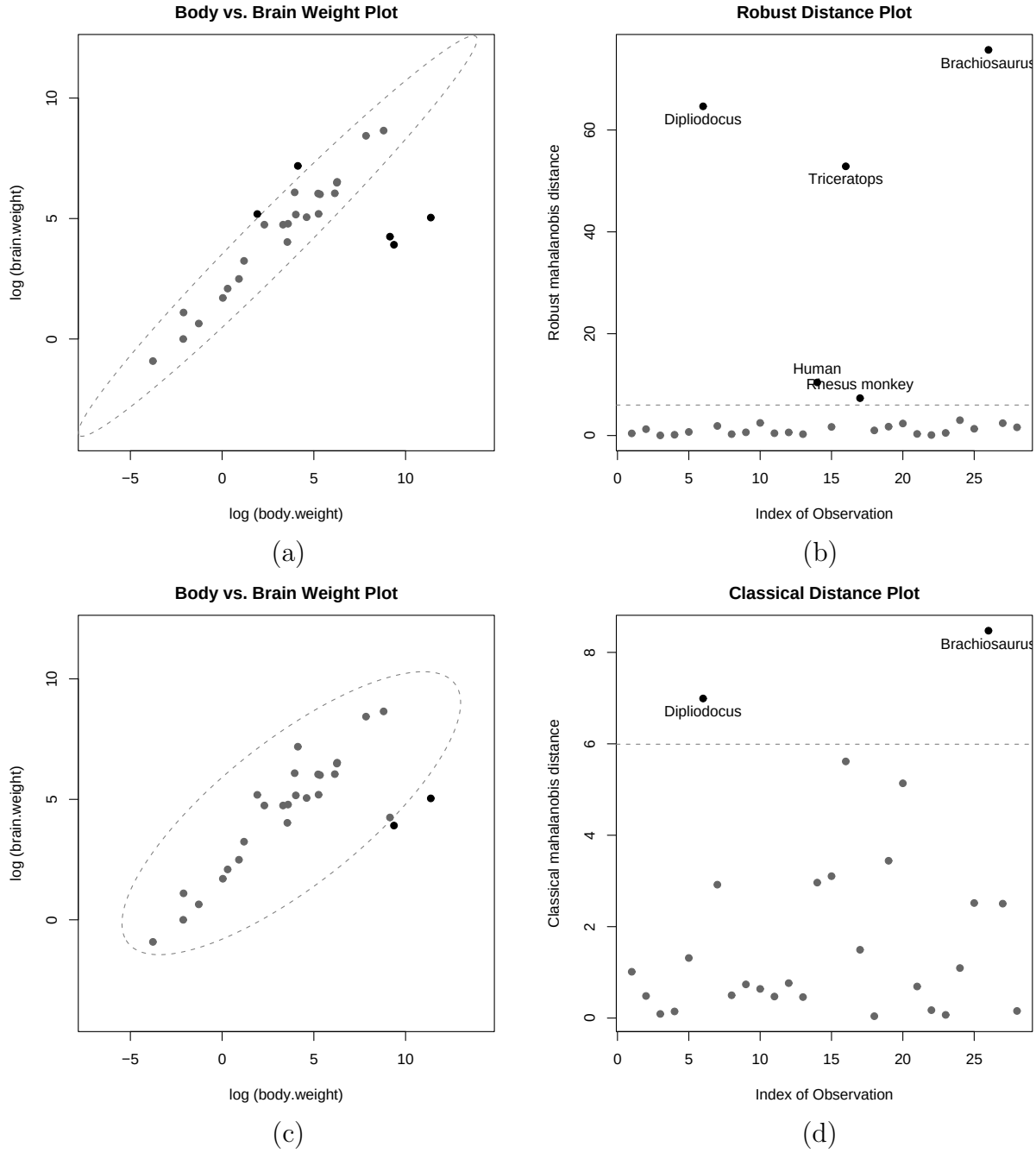


Figure 2.6: Body and brain weights of 28 animals. Source: P. J. Rousseeuw and A. M. Leroy (1987) Robust Regression and Outlier Detection. Wiley, p. 57. (a) Brain- vs. body weight plot with robust classical covariance structure. (b) Robust Mahalanobis distance plot. (c) Brain- vs. body weight plot with classical covariance structure. (d) Classical Mahalanobis distance plot.

Chapter 3

Imputation Methods

3.1 Introduction

Not only identifying observations, which do not fit the general structure of the data is relevant when checking a data set for consistency, but also *repairing* observations may be considered in this context. Let us consider single observations, which have been identified as outlying, and the reason for this irregularity is, for instance, an error in the process generating the data. The simple approach to treat this issue would be to delete these observations. However the loss of information which comes along with this approach is not tenable. A more elaborate idea would be to *correct* these errors in order to use as much information contained in the data set as possible. This is obviously only possible when it can be assured, that the underlying distribution is homogeneous and all observations belong to the same group. When a whole group of outliers has been detected (Figure 2.6 (a)), which behaves different in comparison to the majority of the data, forcing this group into a structure it does not belong to, is obviously not appropriate. Single values of an observation, which have been identified as defective may from now on be considered as missing values. The only thing we know about them is, they have been determined incorrectly and maybe the reason for this error, but usually they do not contain any further useful information.

Since a data matrix with blanked entries cannot be left in this condition, we have to re-estimate these values in order to obtain a complete data set again. For this purpose so called *imputation methods* have been developed (see Little and Rubin, 2002). These methods try to estimate missing values in a way, that they fit the general structure of the data set as good as possible.

However, the main issue which was decisive for the development of imputation methods is the problem of missings, which is always present when dealing with empirical data. The problems which arise in this context are the prerequisites of common statistical methods which are in many cases designed to handle complete datasets only. Without imputation methods the only chance to apply such statistical methods on incomplete data is to delete observations with missings, which is not tenable because of the enormous loss of statistical

power. If a high dimensional data set with p = 100 is considered and an overall, chance of 5% that a single value is missing, the probability that a single observation does not contain any missings is 0.6%. So skipping all observations with missings in such a high dimensional data set would require to drop 994 out of 1000 observations, which is obviously unacceptable. Further, the reason for the data to be missing may be a component of the underlying structure which must not be ignored and then cannot be seen anymore when observations containing missing elements are simply dropped.

3.1.1 Types of Missing Values

Each process which generates data is influenced by circumstances, which may not have been considered when defining or implementing such a process. In such situations values may be either recorded wrong, or simply no data can be acquired. This may occur, when measuring devices fail, when people do not want to give all information required by a survey - there are many reasons which can lead to incomplete datasets, but not all of the resulting missing values can be treated equally. The difference is the reason why this value is missing, with respect to the not observed and observed values. Rubin and Little (2002) categorize missing values in three different types:

- **Missing Completely At Random (MCAR)**
  There is no relation between observed or not observed values, and the probability that a value is missing.
  $$P\left(\boldsymbol{I}_{ij} = 0 | \boldsymbol{X}_{iM}, \boldsymbol{X}_{i\tilde{M}}\right) = P\left(\boldsymbol{I}_{ij} = 0\right) \tag{3.1}$$
  whereas $\boldsymbol{I}_{ij} = 0$ means, that value $\boldsymbol{X}_{ij}$ is missing, $\boldsymbol{X}_{iM}$ stands for the observed, and $\boldsymbol{X}_{i\tilde{M}}$ for missing values of the i-th observation.

- **Missing At Random (MAR)**
  The probability, that a value is missing depends on the observed values.
  $$P\left(\boldsymbol{I}_{ij} = 0 | \boldsymbol{X}_{iM}, \boldsymbol{X}_{i\tilde{M}}\right) = P\left(\boldsymbol{I}_{ij} = 0 | \boldsymbol{X}_{iM}\right) \tag{3.2}$$

- **Not Missing At Random (NMAR)**
  Both, observed and not observed values are related to the chance, that a value is missing. An example would be a survey where people are asked for their income. Usually people with high salary do not willingly give information about their wealth, whereas people with normal or low income would answer this question. So the income may be directly correlated to the probability of a value to be missing, which is a classical example for NMAR.

	Sepal.Length	Sepal.Width
1	NA	3.03
2	NA	3.71
3	5.10	NA
4	4.78	NA
5	5.01	NA
6	NA	2.93
7	NA	3.57
8	5.31	NA
9	NA	3.45
10	5.61	NA

Table 3.1: Additional random observations to Fisher's iris data set with each exactly one missing value.

3.2 Common Imputation Methods

After discussing the different types of missings, we will have a look at methods for estimating missing values in order to be able to apply statistical methods on data matrices which originally contained blanks. Before considering more elaborate algorithms, we will have a look at common imputation methods, their advantages and disadvantages:

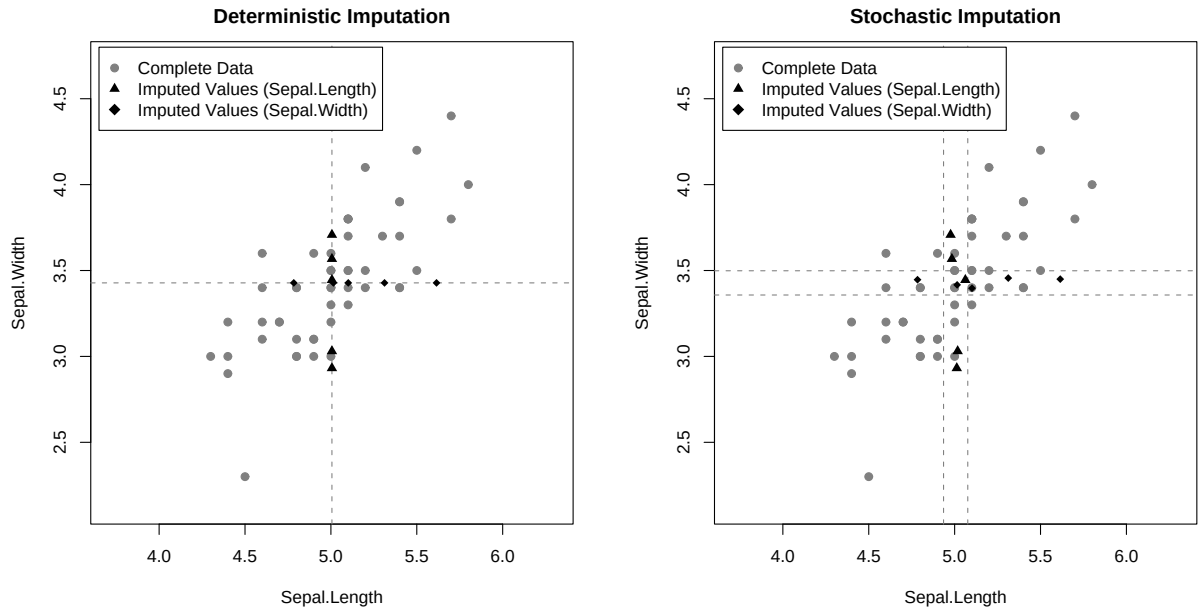
- **Deductive Imputation**

  This method is based on logical dependencies between variables. Some variables may be computed exactly based on others (e.g. age of a person can be calculated when date of birth is known). Since this method requires exactly defined, logical dependencies, which describe the relations between single variables, missings can be restored completely and exactly. A missing value in this context does not mean missing information, so this method cannot be considered as imputation method in its classical sense.

- **Deterministic Imputation**

  The simplest method for imputation is to replace all missing values by a location estimate of the according variable. In this case each column of the data matrix is considered separately, the missing values are removed and the location of the remaining vector is estimated. Afterwards all missings are replaced by this single estimation. The problem of this method is, that replacing all missings with only one value may bias scale estimators, which are strongly affected by the unusual amount of observations exactly in the center of the data.

  Again the iris data set is used to illustrate this approach in Figure 3.1 (a). Artificial observations from Table 3.1 have been added to the data set, whereas each observation has exactly one missing value, coded with *NA*. The values are drawn from a normal distribution with parameters estimated from the original iris data set, again



(a) (b)

Figure 3.1: Fisher's iris data set - only showing observations of type *setosa* with 10 imputed observations each. *Deterministic* (a) and *stochastic* imputation (b).

only considering observations of type *setosa*. The dashed lines represent the center of variables *Sepal.Width* and *Sepal.Length*. Black triangles represent observations where Sepal.Length = *NA*, observations with Sepal.Width = *NA* are represented by black diamonds. Since only one variable is available, the second value is set to the center, so all imputed observations are exactly positioned on one of the two dashed lines, which are given by the center of the data.

- **Stochastic Imputation**

  As an extension of deterministic imputation, *stochastic imputation* adds a random error term $\varepsilon \sim N(0, \sigma_\varepsilon^2)$ to the location estimation. So imputed values do not concentrate that strongly on the center of the data. Since the real location of the data is not known, the imputation is always based on an estimation of the center which is a further issue, usually biasing the resulting structure. A further drawback of these methods is, that they are not intended to be applied on discrete data.

  Figure 3.1 (b) shows an example of stochastic imputation with error term ($\sigma_\varepsilon = 0.01$SD, see Equation (2.8)). Additional observations are again taken from Table 3.1. The dashed lines represent the $2\sigma_\varepsilon$ interval around the center of the data. Observations with missings do here not concentrate exactly on two lines, but are a little bit scattered in between these borders.

  If imputed and original observations from Figure 3.1 would not be represented by different symbols, they would not point out as irregular observations. This, however

is only because of the second principal component, which is quite large in comparison to the first principal component. In Figure 3.2 (a) a data set is shown with high correlation between x and y values. 100 values have been drawn from a multivariate normal distribution:

$$\boldsymbol{X} \sim N_2 \left((4, 3.2), \begin{pmatrix} 1 & 0.8 \\ 0.8 & 0.68 \end{pmatrix} \right) \tag{3.3}$$

x values of observations 81 to 90 and y value of observation 91 to 100 have been blanked and again imputed by stochastic imputation. Since the portion of the second principal component is smaller, three different types of observations point out: The original observations on a diagonal, and the imputed observations on a horizontal and vertical line, depending which variable has been blanked. Since the estimation of missing values does only depend on location information, and is completely independent of any information concerning the covariance structure, deterministic and stochastic imputation are very inefficient since only a fraction of the available information is used, which is a real drawback of these methods. This can be seen in Figure 3.2 (b) which shows the Mahalanobis distances of the original and imputed observations. The dashed line at 2.72 represents the $\sqrt{\chi^2_{2,0.975}}$ - quantile. Only 5 out of 20 imputed observations are below this boundary, whereas only one observation out of 80 original observations is slightly above, which is usual. So 75% of the imputed values can immediately be identified as outliers which is a rather bad characteristic of stochastic and deterministic imputation.

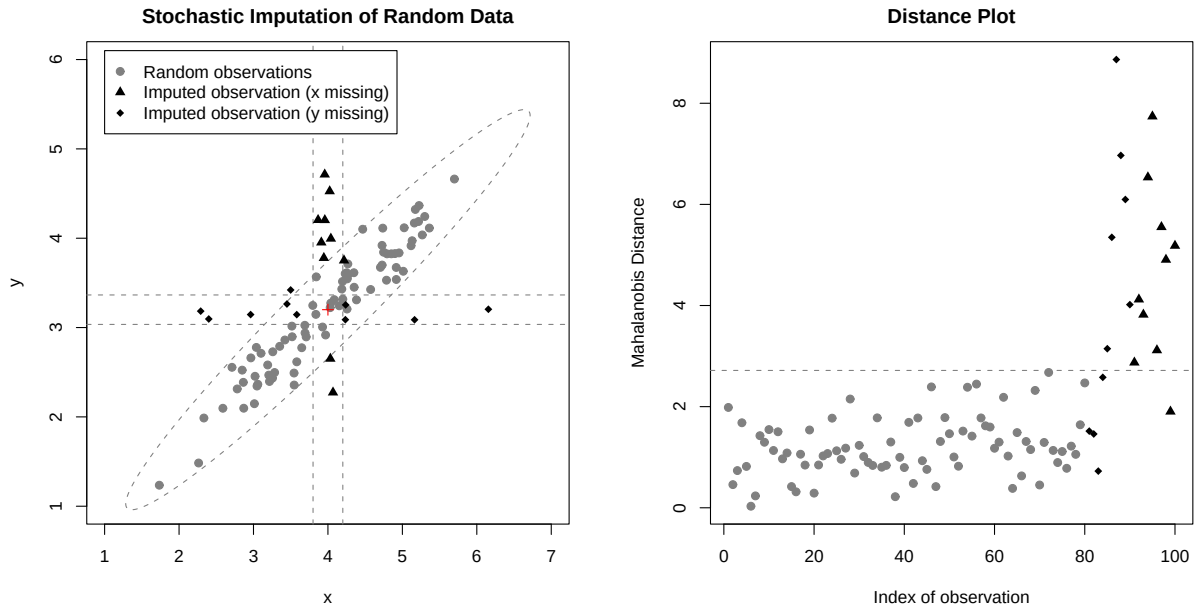
- **Donor Imputation**

  Donor imputation, also known as *Matching*, concentrates on single, complete observations. Estimation of missing values of a specific observation always depends on one complete observation, so imputed values follow the general structure of the data. By considering Mahalanobis-distances, it is impossible to distinguish between original and imputed observations.

  In order to estimate missing values of observations, say $\boldsymbol{x}_k$, this method tries to find the most similar observation $\boldsymbol{x}_l$ without missings, which serves as *donor*. Missing values of $\boldsymbol{x}_k$ will be replaced by the available values of $\boldsymbol{x}_l$. In order not to bias the result, one complete observation must not serve as donor too often. This issue is similar to a problem of deterministic imputation: Imputing the same values repeatedly may create artificial accumulations of data, which do not belong to the actual structure and may influence the result of subsequent methods. Since location estimation is not necessary for this method, but only distances are measured, this approach may also be applied to datasets with discrete values, especially in high dimensional cases.

  The calculation of distances may be a little bit tricky, as categorical and continuous variables are usually mixed up. A simple approach for estimating the distance of two observations containing both types of variables may be defined as follows:

$$d = (\boldsymbol{x}, \boldsymbol{y}) = \sqrt{\sum_{i \in C} (x_i - y_i)^2} + c \sum_{i \in D} nequal\,(x_i, y_i) \tag{3.4}$$



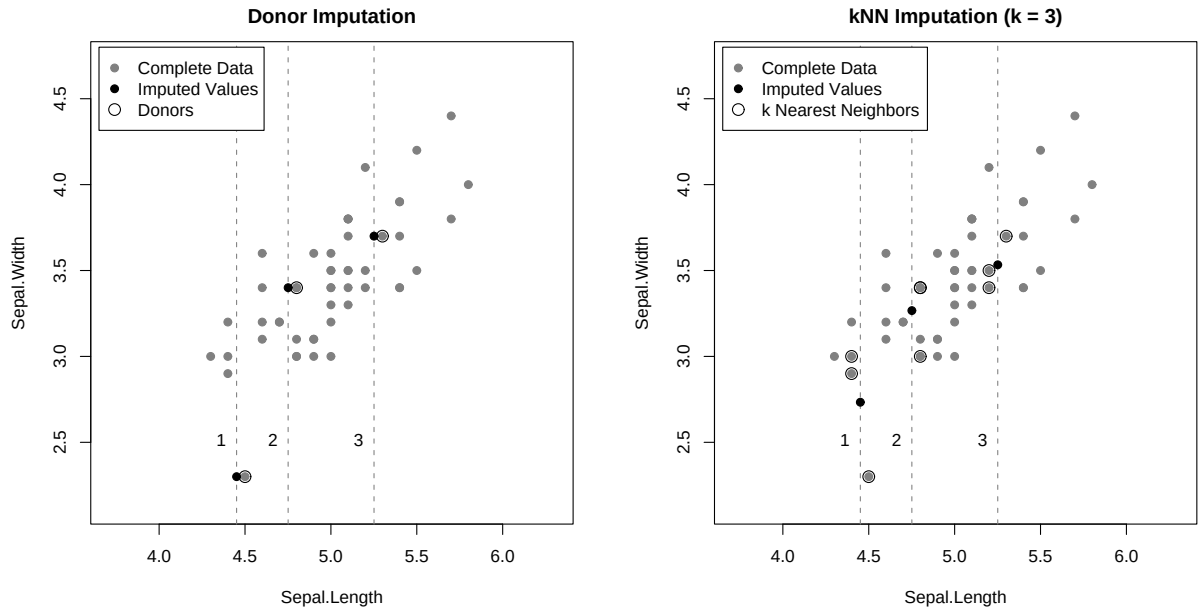
(a) (b)

Figure 3.2: A random sample of 100 multivariate normally distributed two-dimensional observations. 10 x-, and 10 y-values have been blanked and the concerning observations have been imputed by stochastic imputation (a). Distances of orignial, and imputed observations (b).

with C an index set representing continuous variables, D an index set representing discrete variables, and c a constant for adjusting the influence of discrete variables in comparison to continuous variables on the distance estimation. *nequal* returns 0 if the arguments are equal, otherwise 1 is returned.

Figure 3.3 illustrates donor imputation. For this example only three randomly drawn observations have been added with values for *Sepal.Length* = 4.45, 4.75 and 5.25. Values of *Sepal.Width* are missing. Imputed observations are displayed as black dots, observations which serve as donor have been marked additionally. Concerning this example, two problems should be mentioned:

- For the additional observation 1 on the left, a rather outlying observation has been chosen as donor. Since the imputed values of an observation with missings depends on only one complete donor, this method is very instable concerning outliers.
- The second issue concerns the imputed observations 2 and 3. Although a plausible value has been chosen for the variable *Sepal.Width*, there is not only one donor with minimum distance regarding variable *Sepal.Length*. There are several solutions which are equal in terms of goodness, without any chance of finding a *best* solution. The problem arises in this context because the data type of *Sepal.Length* and *Sepal.Width* is not really continuous, but the values have been rounded to



(a) (b)

Figure 3.3: Fisher's iris data set - only showing observations of type *setosa* with three imputed observations each. *Donor* (a) and kNN imputation (b).

tenths of millimeters, so these variables may be considered as discrete under certain circumstances. If the variables were really continuous, or the data set consists of more than two discrete dimensions, the problem of two possible candidates for donors won't be that outstanding. However, apart from outliers, imputed values are obviously located corresponding to the general structure of the data.

- **kNN Methods**

  The *k-Nearest Neighbors* method may be considered as extension of Donor Imputation. Not only one similar observation, but k similar observations are considered in order to estimate missing values. These k observations are referred to as the k Nearest Neighbors. Usually an average of the concerned observations is being calculated and replaces the missing values. An advantage of this algorithm is the adjustable variance, which decreases with increasing k. Further this method is parameter-free, so no distribution assumptions have to be made in advance.

  The kNN-approach is shown in Figure 3.3 (b). In comparison to the previous method, an imputed observation does here not depend on one, but on k donors, so the estimation of missing values is more robust concerning single outliers. The value for *Sepal.Width* of the mentioned observation 1 on the left has now changed, so that it fits quite well into the general structure. Further the chance of obtaining more than one possible solution which is equal in quality is not that high anymore, so there are even less problems with discrete variables in this context.

3.3 Notation, Indexing and Definitions

In the following sections we will often transfer observations and other expressions from one system of coordinates to another. This is usually done with rotation matrices, which is in our case $\mathbf{\Gamma}$, the loadings of the principal components. Since we will primarily consider sub-spaces of our p-dimensional data space, we will refer to parts of matrices which will be denoted in the following way:

$$\boldsymbol{A}_{MK} \quad \boldsymbol{A}_{mk} \tag{3.5}$$

Capital letters in this context always represent index sets. So $\boldsymbol{A}_{MK}$ stands the matrix $\boldsymbol{A}$, only considering rows from index set M and columns identified by index set K. Small letters represent indices, so $\boldsymbol{A}_{mk}$ stands for the k-th element in the m-th row of $\boldsymbol{A}$. Further, whole columns or rows from a matrix may be referred to as

$$\boldsymbol{A}_{.K} \quad \boldsymbol{A}_{M.} \tag{3.6}$$

whereas $\boldsymbol{A}_{.K}$ stands for all columns of $\boldsymbol{A}$ which are given by index set K, and $\boldsymbol{A}_{M.}$ represents the rows, given by index set M. This can be again done with small letters, thus only one column or row is selected. The priority of indexing is higher than any other matrix operation, so

$$\boldsymbol{A}_{.K}^{\top} = \quad (\boldsymbol{A}_{.K})^{\top} \tag{3.7}$$

Since we permanently refer to k relevant components of our data structure, we will often need the first k columns of a matrix so the index sets

$$K := \{1, \ldots, k\} \tag{3.8}$$

and

$$\tilde{K} := \{k+1, \ldots, p\} \tag{3.9}$$

shall here be defined a priory. Further we often split a vector representing an observation into its observed and missing variables, which can easily be done by index sets

$$M_i = \{j = 1, \ldots, p | \boldsymbol{X}_{ij} \neq NA\}\,; \quad m_i = |M_i| \tag{3.10}$$

and the complementary

$$\tilde{M}_i = \{j = 1, \ldots, p | \boldsymbol{X}_{ij} = NA\} = \{1, \ldots, p\} \setminus M_i; \quad \tilde{m}_i = |\tilde{M}_i| \tag{3.11}$$

So M_i indicates variables which values are known in a observation $\boldsymbol{X}_{i.}$, whereas $\tilde{M}_i$ indicates variables with missing values. Further m_i is the number of available values and $\tilde{m}_i$ ist the number of missing values of the considered observation.

Since we will primarily use index sets K and M for $\mathbf{\Gamma}$, where rows represent variables and columns represent principal components, when referring to $\mathbf{\Gamma}$ the '.' for indexing a whole column or row with index sets K, $\tilde{K}$, M or $\tilde{M}$, will be omitted. So $\mathbf{\Gamma}_K$ stands for the **columns** of $\mathbf{\Gamma}$ specified by index set K, whereas $\mathbf{\Gamma}_M$ stands for **rows** of $\mathbf{\Gamma}$ given by M.

Further the p-dimensional identity matrix, which is usually denoted as $\mathbf{I}_p$ will be here displayed as $\mathbf{I}^p$, as with this notation indexing of such a matrix is more comfortable.

3.4 Imputation based on Principal Components

Since imputed values should fit the structure of the majority of the observations as good as possible, and if the observations are from a multivariate normal distribution, this structure is described sufficiently by principal components. Thus they can be used for imputing missing values. The first task which has to be solved is to develop an algorithm for PCA which can be applied on data containing missings. For this purpose Walczak et al. (2001) introduced a method for PCA, which can cope with missing values at random. Serneels et al. (2007) suggested to extend this method by using robust PCA. This approach can be used for imputation purpose as well, since the algorithm iteratively estimates missing values for applying common PCA methods.

3.4.1 PCA on Data Containing Missing Elements

Generally the algorithm repeatedly estimates principal components, projects observations containing missing onto these components and stops when the estimation of the missings does not change anymore. Before going into detail, a matrix $\boldsymbol{I}$ shall be defined with entries

$$\boldsymbol{I}_{ij} = \begin{cases} 1 & \text{if } X_{ij} \text{ observed} \\ 0 & \text{if } X_{ij} \text{ missing} \end{cases} \tag{3.12}$$

which represents the pattern of missing values. Since missings are usually coded and denominated as *NA* (Not Available) this pattern may also be called NA-pattern. The procedure is described as follows:

1. Missing values are initialized with a location estimate or by a donor imputation.

$$\hat{\boldsymbol{X}}_{ij}' := \begin{cases} \boldsymbol{X}_{ij}, & \boldsymbol{I}_{ij} = 1 \\ \text{initial estimate}, & \boldsymbol{I}_{ij} = 0 \end{cases} \tag{3.13}$$

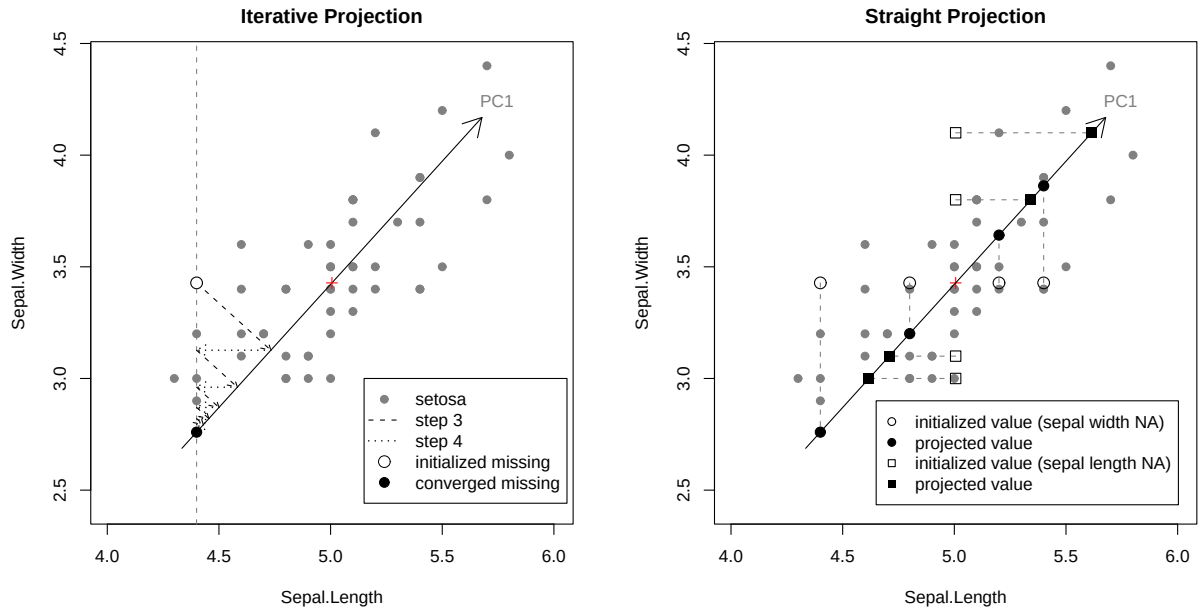
2. The singular value decomposition is performed on $\hat{\boldsymbol{X}}'$. It is mentioned, that PCA can also be applied at this point. Serneels et al. (2007) suggest to use a robust PCA method in order to robustify the result. So eigenvalues $\hat{\boldsymbol{\Lambda}}$ and eigenvectors $\hat{\boldsymbol{\Gamma}}$ of the actual structure are estimated. Scores are calculated in the usual way (see 2.3):

$$\hat{\boldsymbol{Z}} := \hat{\boldsymbol{X}}'\hat{\boldsymbol{\Gamma}} \tag{3.14}$$

3. The original data matrix is reconstructed with k components, $k < p$

$$\hat{\boldsymbol{X}}'' := \hat{\boldsymbol{Z}}_{\hat{K}}\hat{\boldsymbol{\Gamma}}_{\hat{K}}^{\top} \tag{3.15}$$

whereas $\hat{\boldsymbol{Z}}_K$ and $\hat{\boldsymbol{\Gamma}}_K$ denote in this context the first k columns of $\hat{\boldsymbol{Z}}$ and $\hat{\boldsymbol{\Gamma}}$.



(a) (b)

Figure 3.4: Fisher's iris data set - only showing observations of type *setosa*. Comparison of the iterative projection algorithm as proposed by Walczak et al. (2001) (a) and the straight projection method (b).

4. The missing values of the original matrix are replaced by the values from the reconstructed matrix.

$$\hat{\boldsymbol{X}}_{ij} = \begin{cases} \hat{\boldsymbol{X}}_{ij}, & \boldsymbol{I}_{ij} = 1 \\ \hat{\boldsymbol{X}}''_{ij}, & \boldsymbol{I}_{ij} = 0 \end{cases} \tag{3.16}$$

The procedure is continued at step 2 until convergence of the system. Walczak et al. (2001) do not suggest a convergence criterion, but Serneels et al. (2007) suggest to use the difference of the estimated values between two iterations. Other convergence criteria like the movement of the principal components will be described later in Section 4.3

A further cross-validation step is suggested in order to determine the optimal number of components for reconstructing the data matrix. In fact, the number of principal components which should be used, is the number of relevant components contained in the data structure. However, as mentioned in Section 2.3.1, the decision how many components shall be considered when examining a specific set of data is not easy, as there are no formal rules which can be applied on this issue.

Figure 3.4 (a) illustrates the *movement* of imputed observations from their initial estimation to their final position in converged state. Dashed lines represent step 3, which projects the observation onto the principal component. However this projection is orthogonal to the estimated principal components and not to the original system of coordinates which implies

an alteration of observed values, which of course is illegitimate. In the mentioned Figure an observation with missing information of *Sepal.Width* is intimated by the dashed vertical line. An imputed value must always be located exactly on this line, which here is the case with the initial estimate.

The projection onto the first principal component not only changes the value of *Sepal.Width* but *Sepal.Length* is also affected. So step 4, represented by the dotted line, has to correct this deviation from the dashed vertical line, which is simply done by only adopting values of *Sepal.Width*. With iteration #2 the observation will be again projected onto the principal component and then moved back onto the dashed line. The only difference to iteration #1 is, that meanwhile the principal components have been estimated again, and since some observations have changed their positions, the principal components have moved as well. For the sake of simplicity Figure 3.4 (a) refers to a situation, where the moving observation does not have any influence on the principal components, which may be the case when the considered observation is outlying and a robust PCA algorithm is applied.

Figure 3.4 (b) shows how several observations would be projected onto the first principal component. Observations with missing values of *Sepal.Width* are displayed as white circles, observations without information concerning *Sepal.Length* are represented by white squares. Black items represent again the converged state of imputed observations. The situation displayed here is similar to prediction with linear models, where one or more dependent variables are explained by one ore more independent variables. The difference here is, that during one iteration each variable may become dependent or independent - depending on the NA-pattern of the currently considered observation.

Since the algorithm must not change observed values, and there may be different NA-patterns for each variable, the direction of the projection is very likely to be different for different observations, especially when considering high dimensional matrices. In a two dimensional case there are obviously two possible directions for projecting single observations (parallel to the ordinate or parallel to the abscissa), but in p-dimensional space there are $2^p - 2$ possible directions. This results from 2^p possible NA-patterns for each observation, minus 2 for the trivial cases which do not require projection: No value is missing, so no imputation is necessary, and all values are missing, which is a rather unusual situation and may be treated by deleting such observations, or as no information is available, only deterministic, or stochastic imputation may be applied in such cases.

Convergence

Serneels et al. (2007) points out, that robust PCA methods may have problems with convergence because of certain observations which prevent the algorithm from termination.

As convergence criterion Serneels et al. (2007) consider the movement of an imputed value between two iterations. In some cases the estimated values for missing values may differ strongly between two iterations. Serneels et al. (2007) try to explain this behaviour by dividing the observations containing missings into two groups: *normal* and *not normal* data

points which are in fact outlying and not outlying observations. Since robust PCA does not consider outlying observations, estimation for such observations with missings may alter strongly between two iterations. To bypass this problem the convergence criterion, which measures the movement of estimations of missing values between single iterations should only consider observations, which do not exceed a distance threshold of $\sqrt{\chi^2_{df,0.99}}$.

The explanation of Serneels et al. (2007) only considering this problem in relation to robust PCA methods may not be sufficient, as using classical PCA in this context may cause similar problems. Observations with high Mahalanobis distance would still make enormous movements, even when the components do hardly move anymore. So a small movement of principal components which can occur even in (nearly) converged state may prevent the algorithm from termination because of some outlying observations containing missings and breaking the convergence criterion. So the $\sqrt{\chi^2_{df,0.99}}$ boundary for the distance measurement between observations may even be considered when using classical PCA.

3.4.2 Estimating Concrete Values

Walczak et al. (2001) and Serneels et al. (2007) originally intended to estimate principal components, but since an estimation of missings values is done anyhow, the proposed algorithm actually does the whole job. Since a corresponding implementation of the mentioned method would only return PCA-information, slight changes have to be done in order to retrieve the estimated values as well, which usually can be done very easily by modifying the return structure of the corresponding method.

Chapter 4

Improvement of Existing Imputation Methods

In this chapter we will try to adopt the algorithm, proposed by Walczak et al. (2001), and Serneels et al. (2007), in order to develop a new imputation method. This original algorithm will from now on be referred to as the iterative algorithm. Since they did not intent to develop an algorithm for imputation, but an algorithm for (robust) PCA on data containing missings, several issues arise, which have not been mentioned in their publications yet. Improvement of the existing algorithm considering PCA is not possible, but the quality of imputation can be increased in several situations.

In order to increase the quality of the existing algorithm with respect to imputation, we will consider the following aspects:

- **Projection Algorithm**

  We will have a closer look at the projection of observations containing missings onto the principal components. As alternative to the iterative projection algorithm illustrated in Figure 3.4 (a) we will find a straight way of projecting these observations like shown in Figure 3.4 (b).

- **Convergence Criterion**

  The convergence behaviour of the existing algorithm will be considered, and alternatives will be discussed. At the moment convergence is measured depending on the movement of imputed observations. We will see, when using different projection methods, convergence of the algorithm can also be measured based upon the movement of the principal components.

- **Runtime**

  A critical aspect concerning robust algorithms is always the computational effort, which usually exceeds the effort of classical estimates by far. Since many robust algorithms are based on sort mechanisms, which are computationally expensive, trying to find a

way of reducing the number of iterations of an algorithm would be very helpful in this context.

- **Quality Criterion**

  In order to compare different imputation algorithms, it will be necessary to formulate criteria which represent the quality of an estimation. These criteria may depend on the deviation of missing observations from their original value or may be based on the deviation of estimated principal components from the original components. Usually original values and components are not known, so the only way to estimate the quality of different algorithms in this way is to set up a simulation study. A complete data set is chosen and a certain portion of the values is blanked randomly. Afterwards the imputation algorithm tries to re-estimate the blanked values, and these estimates can then be compared to the original values.

4.1 Types of Projections

The issue treated here is the improvement of step 3 and 4 from Section 3.4.1, the projection of observations with missing values on a hyperplane, spanned by several principal components. The iterative projection of the data is unnecessary since all information concerning the projection is available at the first iteration. The only difference between projecting the components in one step like shown in Figure 3.4 (b) from projecting them iteratively (a) is, that during iterative projection the components move, and so the result may be a little bit different. However, we will show that this difference can be ignored in most cases.

Before being able to formalize projection algorithms, we must have a closer look at different types of projections which can occur when imputing missing values based on principal components. We have already figured out, that we do not have only one direction the data is being projected (e.g. Figure 3.4 (b): parallel to *Sepal.Width* vs. parallel to *Sepal.Length*) but the projection itself depends on the NA-pattern. Considering the projection, observations with the same NA-pattern can be treated equally, so we actually categorize types of NA-patterns, according to their number of missings values and the number of relevant principal components in the system.

The projection of observations in the two dimensional case ($p = 2$) is done along a straight line, which is defined by the available values. Such lines are shown in Figure 3.4 (a). Since only the value of *Sepal.Length* is known, the observation has to be located somewhere on the corresponding vertical line ($m = \tilde{m} = 1$). We assume, that the displayed data set has one relevant principal component ($k = 1$). So there is exactly one point, where the first principal component and the vertical line cross, which is marked by the black dot. At this point the observation will be positioned when the algorithm of Walczak et al. (2001) has converged.

In high dimensional cases it is not always that simple, as already intimated by the huge amount of possible directions for projecting the data. In order to keep the explanations as

simple as possible we will from now on focus on one specific observation $\boldsymbol{x} := \boldsymbol{X}_i$ and the according NA-pattern, so the index of M_i, $\tilde{M}_i$, and so forth will be omitted.

Further two hyperplanes shall be mentioned, which will be important from now on:

$$\Phi(\boldsymbol{\gamma}) = \boldsymbol{T} + \boldsymbol{\Gamma}_K \boldsymbol{\gamma}, \tag{4.1}$$

whereas $\boldsymbol{T}$ is a location estimate of the data, and $\boldsymbol{\Gamma}_K$ are the first k columns of the loadings $\boldsymbol{\Gamma}$. $\boldsymbol{\gamma}$ is any vector $\in R^k$. When iterating through each possible value of $\boldsymbol{\gamma}$, each point on the hyperplane, which is spanned by the first k principal components, is generated. In the example shown in Figure 3.4, Φ is identical with the first principal component.

$$\Psi_x(\boldsymbol{\eta}) = \mathbf{I}^p_{\cdot M} \boldsymbol{x}_M + \mathbf{I}^p_{\cdot \tilde{M}} \boldsymbol{\eta} \tag{4.2}$$

This hyperplane of dimension $\tilde{m}$ is given by the observation $\boldsymbol{x}$. It covers all points where an imputed observation may be located without changing observed values $\boldsymbol{x}_M$. $\mathbf{I}^p_{\cdot P}$, in general, is an identity matrix of size $p \times p$ but only using columns contained by index set P. So what $\Psi_x(\boldsymbol{\eta})$ actually does, is to combine the observed values from $\boldsymbol{x}$ and an estimation of missing values $\boldsymbol{\eta}$ to one vector in the appropriate order.

For example an observation

$$\boldsymbol{x}^\top = (NA, 1, NA, 2, 3, NA, NA)$$

can then be considered as a hyperplane $\Psi_x(\boldsymbol{\eta})$, whereas $\boldsymbol{\eta}$ represents missing values. So by only specifying $\boldsymbol{\eta}$, we again obtain a vector which represents a complete observation:

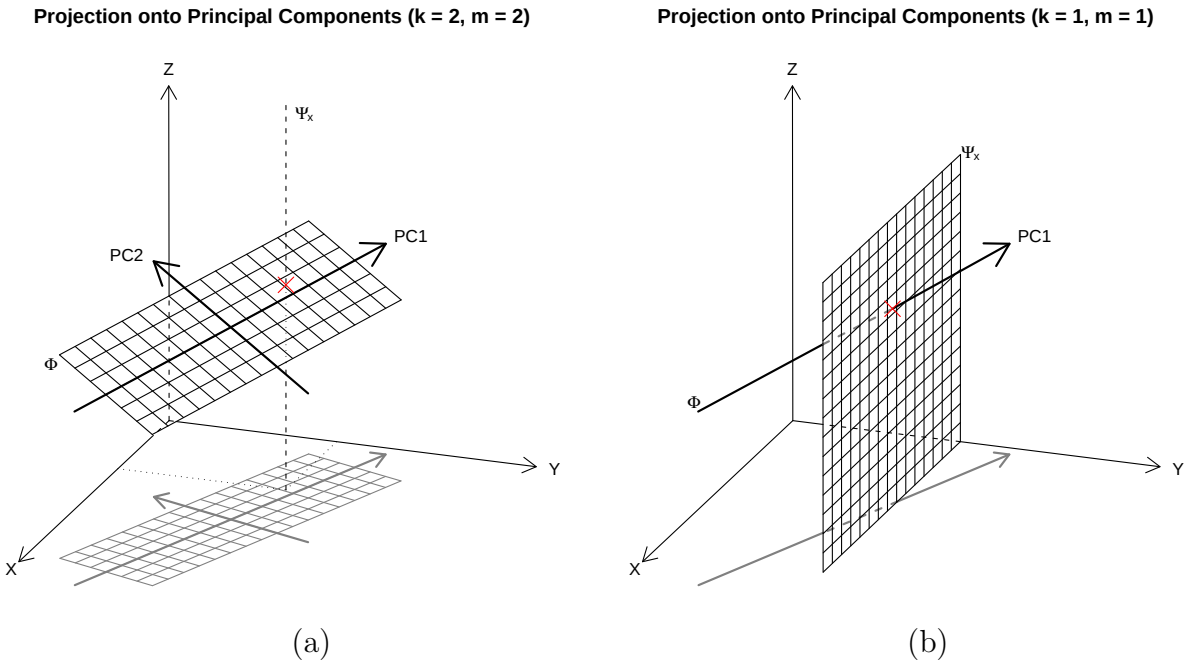
$$\Psi_x(9, 8, 7, 6) = (9, 1, 8, 2, 3, 7, 6)$$

Such a formal definition, which only merges two vectors may seem dispensable, but in future we will treat observed and missing values as separated vectors. So it will be very helpful to have a mathematical expression merging these vectors back to a complete observation which then can be rotated and shifted like a complete observation.

Since the direction of Ψ only depends on the NA-pattern of an observation, such hyperplanes of observations with the same NA-pattern are always parallel. Observed values only specify the location of this hyperplane. Further the direction of this hyperplane is always different for observations with different NA-patterns. An example for Ψ is given in Figure 3.3 (b), where all dashed lines are such hyperplanes. Since this is only a two dimensional example and $m = 1$, these hyperplanes are one dimensional and thus lines.

When considering a higher-dimensional data set, it is not always that simple. At first we will consider all possible constellations in three dimensional space, and then extend these cases to p dimensional spaces. Figure 4.1 shows principal components and observations containing missings in three dimensional space, and demonstrates two possibilities of projection.

- $k = 2$, $m = 2$



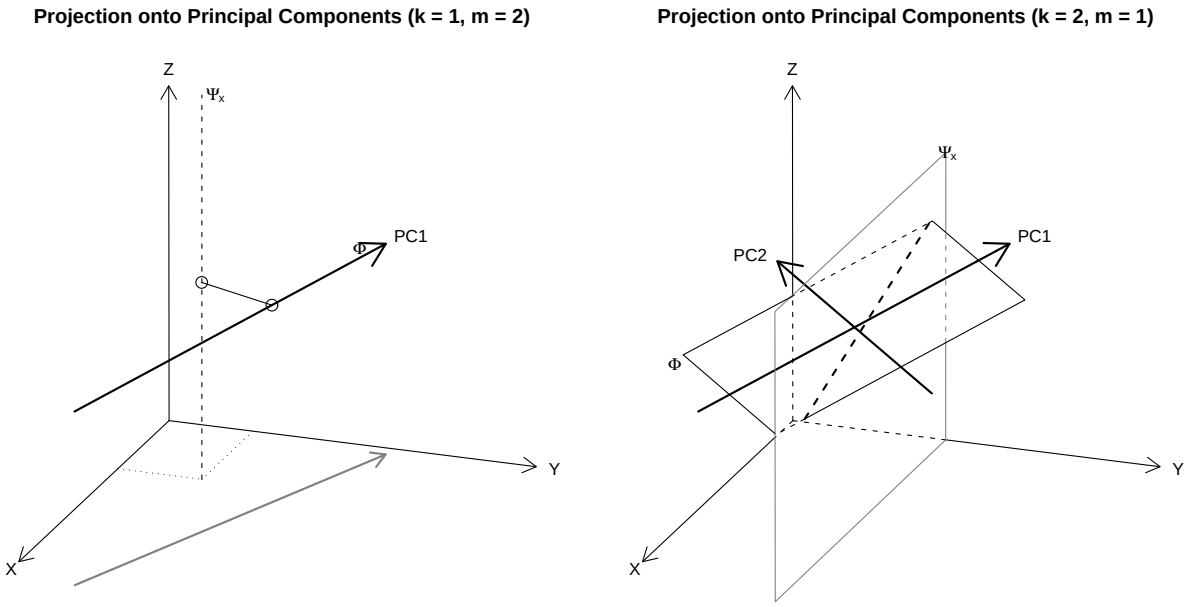
(a) (b)

Figure 4.1: Schematic illustration of the intersection of Ψ_x and Φ. Observations are omitted but concentrated on a plane (a) and one principal component (b). Since the considered type of projection delivers an exact and unambiguous solution, the intersection of Φ and Ψ_x is a point, marked as cross. The considered observation is displayed as line (a) and as grid (b).

In this case we consider an observation which has one missing value, which here is the z-variable. This observation is intimated by the vertical, dashed line in Figure 4.1 (a), which is given by the known values of x and y. Further this line represents the hyperplane Ψ_x. In this example we assume, that our data set has two relevant principal components, represented by $PC1$ and $PC2$ in this plot. The grid is a plane, which is spanned by these components and so illustrates Φ. Since we know, that our data set is scattered among the first two principal components, and we are searching for an observation which is located Ψ_x, it is obvious, to select the intersection of Ψ_x and Φ. This point is marked by a $\times$ in both plots.

- $k = 1$, $m = 1$

  Another possibility, demonstrated in Figure 4.1 (b) would be, that all observations are mainly scattered around one principal component which would then be equal to Φ, and we consider an observation which consists of two missings and one available value, which here are x and z missings and y known. So the imputed observation will be located anywhere on a plane which is parallel to the plane orthogonal to the y-axis with distance y from the center. This plane is represented by Ψ_x in this context, and displayed as the vertical grid. Again we have exactly one point, where Ψ_x and Φ intersect. Since our observation is locally forced onto Ψ_x, and we know, that it is most likely, that an observation occurs exactly on Φ, we again choose the point of



(a) (b)

Figure 4.2: Schematic illustration of projecting an observation onto Φ. Φ and Ψ_x are both one-dimensional, they do not intersect in three dimensional space (a). Φ and Ψ_x are both two-dimensional, the intersection is displayed as thick, dashed line (b). Observations are omitted.

intersection for the location of our observation.

In this context cases with all or no values missing ($m = 0$ or $m = p$) are irrelevant, since no projection or imputation can be done. So two further possibilities are left, which are mixtures of the preceding situations:

- $k = 1$, $m = 2$

  Figure 4.2 (a) illustrates an example where again only one principal component explains a relevant amount of variance, and an observation with only one missing value, so Ψ_x, represented by the dashed line, and Φ, the first principal component, are both one-dimensional.

  The first problem regarding our projection model shows up, as Φ and Ψ_x do not intersect at all. In this situation we have to make a compromise. Since our observation is forced onto Ψ_x, but it is more likely, that an observation occurs anywhere on Φ, we have to search two points on the hyperplanes with minimum distance. Here these points are shown as $\circ$ at the ends of the line connecting these hyperplanes. Though the lines in the plot may look like they are intersecting in one point, they have a certain distance to each other. The projection of the first principal component onto the "floor" which is the plane spanned by the x- and y-axis should make it clear, that those lines are not intersecting at any point. Of course in certain cases Φ and Ψ_x may have a

point of intersection, but the chance that the principal component hits Φ exactly is zero when real continuous data are considered.

- $k = 2$, $m = 1$

  The fourth, and last possibility in the three-dimensional space is, that the majority of variance is explained by two principal components, and an observation contains two missing values. This situation is shown in Figure 4.2 (b). Ψ_x represented by the grey bordered area, as well as Φ, displayed as black bordered area, are both two-dimensional. So the intersection of these hyperplanes is now a line, and no more a single point. So the projection of our observation onto the principal components delivers an ambiguous solution. In this case, the observation may be located anywhere on the resulting line, which is displayed thick and dashed in the corresponding plot.

  Since the approach of projecting observations onto principal components is no more sufficient in this case, we will have to find another criterion, which identifies the best of all possible solutions. This criterion will be discussed in Section 4.2.3 and basically depends on the Mahalanobis distance.

Since up to this point only two and three dimensional spaces have been considered, the preceding approach for categorizing different projection types shall now be generalized to p-dimensional data sets:

- $m = k \Leftrightarrow k + \tilde{m} = p$

  The number of relevant principal components plus the number of missing values equals p, the dimensionality of the data set, so in most cases the system can be solved exactly. If Φ and Ψ_x intersect, there is exactly one point of intersection. Still it is possible, that these hyperplanes are parallel and they do not intersect, but this means, that at least one principal component has to be exactly parallel to an axis, or the space spanned by several axes of the original system of coordinates, which is rather unusual.

- $m < k \Leftrightarrow k + \tilde{m} < p$

  In this case, $dim\,(\Phi) + \dim\,(\Psi_x)$ is smaller than p, which means that the hyperplanes do usually not intersect, so the system is over-determined. However there is a chance, that the planes have a point of intersection, but this chance is practically zero (Ψ_x and Φ in Figure 4.2 (a) would have to intersect). In this case we search a point in Ψ_x which has minimum distance to Φ.

- $m > k \Leftrightarrow k + \tilde{m} > p$

  Here $dim\,(\Phi) + \dim\,(\Psi_x)$ is greater than p. We get more than one solution of our problem, and so the system is under-determined. In Figure 4.2 (b) all possible solutions are located on the thick, dashed line. Generally we can say, that the solution obtained with this approach is a third hyperplane Ω of dimension $k + \tilde{m} - p$. An unambiguous solution for this case will be developed later, when exactly specifying the projection algorithms.

For the sake of completeness, it should be mentioned, that even this case may not deliver any solution, since Ψ_x and Φ could be parallel, which again is very unlikely. This would imply, that principal components are parallel to the axes of the original system of coordinates, which only occurs, when there is no covariance between the single variables of $\boldsymbol{X}$ at all ($\boldsymbol{C}$ is diagonal). Since there would be no relation between the variables, in such a case an estimation or imputation of a variable based on other variables is not possible at all, so all imputation methods based on the covariance structure would fail. The only possibility in this case would be to apply stochastic or deterministic imputation methods, which are, as discussed in Section 3.2 not very powerful.

4.2 Projection Methods

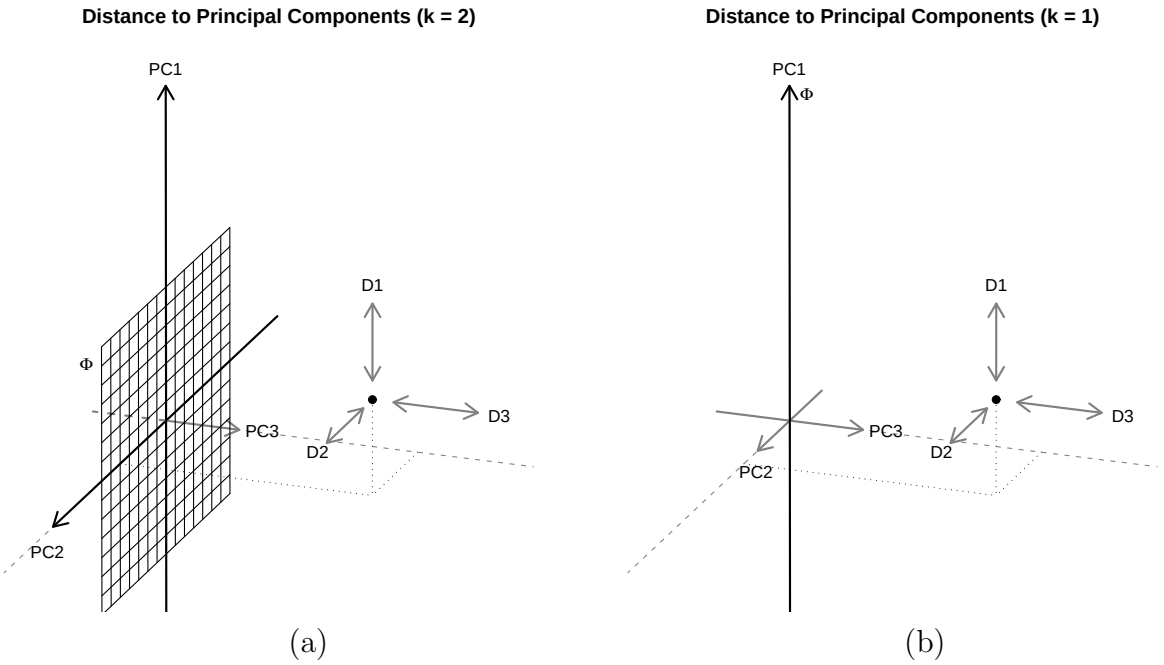
After discussing different situations which can occur when projecting observations containing missing values onto k principal components, and specifying Φ and Ψ_x, two importatnt subspaces in this context, we will find a way of calculating the point of intersection of these hyperplanes, if possible, and if not possible we will develop alternative approaches in order to find an appropriate solution.

4.2.1 Over-Determined Systems

We start with a rather complicated situation, which occurs when Φ and Ψ_x do not intersect. As already discussed, this case appears when $k+\tilde{m} < p$. Since our observation is forced onto Ψ_x, but we want to locate it as near to Φ as possible, we will find a point $\in \Psi_x$ with minimum distance to Φ. What has to be done now, is to develop an expression which represents the distance of a point $\in \Psi_x$, depending on $\boldsymbol{x}_{\tilde{M}}$ which then can be minimized.

As we want to keep the model for explaining those relations as simple as possible, we will now consider the situation in the system of the principal components. From this point of view the principal components are equal to the axes of the system. Φ is simply a hyperplane which is spanned by the first k axes. This is shown in Figure 4.3 (a) and (b). The principal components are equal to the axes of the system. In Figure 4.3 (a) $dim\,(\Phi) = 2$, in Figure 4.3 (b) $dim\,(\Phi) = 1$. The original system of coordinates is here omitted, since it is not relevant in this context.

From this point of view it is now very easy to measure the distance of an observation from Φ. Since Φ is orthogonal to the system of coordinates, a movement of an observation in the direction of one of the first k principal components does not change its distance to Φ. In Figure 4.3 an observation is displayed as black dot, and three arrows intimate a movement into the directions of the principal components. When the observation is moved in direction $D1$ or $D2$, the movement would be parallel to the grid, which represents Φ, and no change of distance would occur. However when the observation moves in direction $D3$, its movement is orthogonal to Φ, and the distance is influenced. In this example only the distance in



(a) (b)

Figure 4.3: Distances of observations from Φ with $dim\,(\Phi) = 2$ (a), and $dim\,(\Phi) = 1$ (b).

direction $D3$ which is the value of the third principal component is relevant. When rotating a centered observation $\boldsymbol{x} = (x_1, x_2, x_3)$ into the system of principal components

$$\boldsymbol{x}^{(pc)} = \boldsymbol{\Gamma}^\top \boldsymbol{x} \tag{4.3}$$

the distance from Φ would simply be $\left|x_3^{(pc)}\right|$.

Figure 4.3 (b) shows a similar situation, but here the dimensionality of Φ has been reduced to 1, so there is only one principal component available. In this case a movement of the observation in direction $D2$ and $D3$ influences the distance from Φ, thus we will have to consider the amount of the second and third component for distance calculation.

$$d^2 = \left(x_2^{(pc)}\right)^2 + \left(x_3^{(pc)}\right)^2 \tag{4.4}$$

When extending this approach to p dimensions, we have to calculate the Euclidean distance of observation $\boldsymbol{x}^{(pc)}$ from the origin on a hyperplane, spanned by principal components $k + 1, \ldots, p$:

$$d^2\left(\boldsymbol{x}^{(pc)}\right) = \sum_{i=k+1}^{p} \left(x_i^{(pc)}\right)^2 = \left(\boldsymbol{x}_{\tilde{K}}^{(pc)}\right)^\top \left(\boldsymbol{x}_{\tilde{K}}^{(pc)}\right) \tag{4.5}$$

We now have found a way of calculating the distance of an observation to Φ, which has been centered and transformed into the system of principal components. In order to find the point $\in \Psi_x$ with minimum distance to Φ we express Ψ_x in the system of principal components:

$$\Psi_x^{(pc)}(\boldsymbol{\eta}) = \boldsymbol{\Gamma}^\top \Psi_x(\boldsymbol{\eta}) \tag{4.6}$$

$$= \mathbf{\Gamma}^\top \left(\mathbf{I}^p_{.M} \boldsymbol{x}_M + \mathbf{I}^p_{.\tilde{M}} \boldsymbol{\eta} \right)$$
$$= \mathbf{\Gamma}^\top \mathbf{I}^p_{.M} \boldsymbol{x}_M + \mathbf{\Gamma}^\top \mathbf{I}^p_{.\tilde{M}} \boldsymbol{\eta}$$

(see (4.2))and since $\mathbf{I}^p_{.P}$ simply copies the columns from the preceding matrix which are contained in index set P:

$$\Psi_x^{(pc)}(\boldsymbol{\eta}) = \mathbf{\Gamma}_M^\top \boldsymbol{x}_M + \mathbf{\Gamma}_{\tilde{M}}^\top \boldsymbol{\eta} \tag{4.7}$$

whereas $\boldsymbol{x}_M$ are the observed values of observation $\boldsymbol{x}$, and $\boldsymbol{\eta}$ again represents the missing values. By choosing $\boldsymbol{\eta}$, so that $d\left(\Psi_x^{(pc)}(\boldsymbol{\eta})\right)$ is minimized, we obtain a point $\in \Psi_x$ with minimum distance to Φ.

$$\boldsymbol{\eta}^* = \underset{\eta}{\operatorname{argmin}} \, \mathrm{L}(\boldsymbol{\eta}) \tag{4.8}$$
$$L(\boldsymbol{\eta}) = d^2\left(\Psi_x^{(pc)}(\boldsymbol{\eta})\right)$$
$$= \Psi_x^{(pc)}(\boldsymbol{\eta})_{\tilde{K}}^\top \Psi_x^{(pc)}(\boldsymbol{\eta})_{\tilde{K}}$$
$$= \left(\mathbf{\Gamma}_M^\top \boldsymbol{x}_M + \mathbf{\Gamma}_{\tilde{M}}^\top \boldsymbol{\eta}\right)_{\tilde{K}}^\top \left(\mathbf{\Gamma}_M^\top \boldsymbol{x}_M + \mathbf{\Gamma}_{\tilde{M}}^\top \boldsymbol{\eta}\right)_{\tilde{K}}$$
$$= \left(\boldsymbol{x}_M^\top \mathbf{\Gamma}_M + \boldsymbol{\eta}^\top \mathbf{\Gamma}_{\tilde{M}}\right)_{\tilde{K}} \left(\mathbf{\Gamma}_M^\top \boldsymbol{x}_M + \mathbf{\Gamma}_{\tilde{M}}^\top \boldsymbol{\eta}\right)_{\tilde{K}}$$
$$= \left(\boldsymbol{x}_M^\top \mathbf{\Gamma}_{M\tilde{K}} + \boldsymbol{\eta}^\top \mathbf{\Gamma}_{\tilde{M}\tilde{K}}\right) \left(\mathbf{\Gamma}_{M\tilde{K}}^\top \boldsymbol{x}_M + \mathbf{\Gamma}_{\tilde{M}\tilde{K}}^\top \boldsymbol{\eta}\right)$$
$$= \boldsymbol{x}_M^\top \mathbf{\Gamma}_{M\tilde{K}} \mathbf{\Gamma}_{M\tilde{K}}^\top \boldsymbol{x}_M + \boldsymbol{x}_M^\top \mathbf{\Gamma}_{M\tilde{K}} \mathbf{\Gamma}_{\tilde{M}\tilde{K}}^\top \boldsymbol{\eta} + \boldsymbol{\eta}^\top \mathbf{\Gamma}_{\tilde{M}\tilde{K}} \mathbf{\Gamma}_{M\tilde{K}}^\top \boldsymbol{x}_M + \boldsymbol{\eta}^\top \mathbf{\Gamma}_{\tilde{M}\tilde{K}} \mathbf{\Gamma}_{\tilde{M}\tilde{K}}^\top \boldsymbol{\eta}$$
$$\frac{\partial}{\partial \boldsymbol{\eta}} = 2\mathbf{\Gamma}_{\tilde{M}\tilde{K}} \mathbf{\Gamma}_{M\tilde{K}}^\top \boldsymbol{x}_M + 2\mathbf{\Gamma}_{\tilde{M}\tilde{K}} \mathbf{\Gamma}_{\tilde{M}\tilde{K}}^\top \boldsymbol{\eta} := 0 \tag{4.9}$$
$$\mathbf{\Gamma}_{\tilde{M}\tilde{K}} \mathbf{\Gamma}_{\tilde{M}\tilde{K}}^\top \boldsymbol{\eta} = -\mathbf{\Gamma}_{\tilde{M}\tilde{K}} \mathbf{\Gamma}_{M\tilde{K}}^\top \boldsymbol{x}_M$$
$$\hat{\boldsymbol{\eta}} = -\left(\mathbf{\Gamma}_{\tilde{M}\tilde{K}} \mathbf{\Gamma}_{\tilde{M}\tilde{K}}^\top\right)^{-1} \mathbf{\Gamma}_{\tilde{M}\tilde{K}} \mathbf{\Gamma}_{M\tilde{K}}^\top \boldsymbol{x}_M = \boldsymbol{A}\boldsymbol{x}_M \tag{4.10}$$

We finally have obtained a projection matrix $\boldsymbol{A}$, which transforms the observed values $\boldsymbol{x}_M$ into the missing values $\boldsymbol{x}_{\tilde{M}} = \hat{\boldsymbol{\eta}}$, and so finds the point $\in \Psi_x$ with minimum distance to Φ. Since $\boldsymbol{A}$ does not depend on observed values, but only on $\mathbf{\Gamma}$ and the index vectors M, $\tilde{M}$ and $\tilde{K}$, it has to be calculated for each NA-pattern only once.

This approach can also be used when $k + \tilde{m} = p$ which is the case, when Φ and Ψ have exactly one point of intersection. So the minimum distance of these hyperplanes must be exactly zero in one point, which obviously is the point with minimum distance to each of the hyperplanes, and so is returned by this projection. However the calculation of the point of intersection in this case can be simplified as shown in the next section.

4.2.2 Exactly Determined Systems

This situation is the most comfortable one, since we know that if Ψ_x and Φ intersect, the system delivers exactly one solution. We want to find a point $\boldsymbol{\psi} \in \Psi_x$, which is also $\in \Phi$, so the distance of $\boldsymbol{\psi}$ to Φ is zero. When an observation is located on the hyperplane spanned by the first k principal components, the scores of the principal components ($k +$

$1, \ldots, p$) have to be zero. Since we have already developed the term $\Psi_x^{(pc)}(\boldsymbol{\eta})$ (Equation (4.7)), which represents all possibilities of imputing an observation $\boldsymbol{x}$, in the system of principal components, we only have to set its last $p-k$ values, indexed by $\tilde{K}$, to zero:

$$\left(\Psi_x^{(pc)}(\boldsymbol{\eta})\right)_{\tilde{K}} = \mathbf{0} \tag{4.11}$$

$$\boldsymbol{\Gamma}_{M\tilde{K}}^{\top}\boldsymbol{x}_M + \boldsymbol{\Gamma}_{\tilde{M}\tilde{K}}^{\top}\boldsymbol{\eta} = \mathbf{0}$$

$$\boldsymbol{\Gamma}_{\tilde{M}\tilde{K}}^{\top}\boldsymbol{\eta} = -\boldsymbol{\Gamma}_{M\tilde{K}}^{\top}\boldsymbol{x}_M$$

$$\boldsymbol{\eta} = -\left(\boldsymbol{\Gamma}_{\tilde{M}\tilde{K}}^{\top}\right)^{-1}\boldsymbol{\Gamma}_{M\tilde{K}}^{\top}\boldsymbol{x}_M = \boldsymbol{A}\boldsymbol{x}_M \tag{4.12}$$

Again we have determined the projection matrix $\boldsymbol{A}$ which can be used to project an observation containing missings onto Φ. Since $\boldsymbol{A}$ only depends on principal components and the NA-pattern, it has to be calculated only once for each NA-pattern, and can then be used for all variables according to this pattern.

As this situation is a special case of the over-determined system which has been considered above, it might be interesting comparing the projection matrix $\boldsymbol{A}$ from Equation (4.10) and from Equation (4.12): The only difference which arises from minimizing the distance is, that in (4.10) the term $\boldsymbol{\Gamma}_{M\tilde{K}}^{\top}$ is extended by $\boldsymbol{\Gamma}_{\tilde{M}\tilde{K}}$.

4.2.3 Under-Determined Systems

The third, and most complicated situation occurs, when Φ and Ψ_x intersect, but the intersection is not a point, but again a hyperplane Ω of dimension $w := k + \tilde{m} - p$. This happens, when $k + \tilde{m} > p$. A solution to this issue is to choose a point $\in \Omega$ with minimal Mahalanobis distance in Φ. Since the mathematical formulation of this construction is rather complex, we will formulate an approximation to this approach which will then be evaluated by simulation.

Figure 4.4 illustrates the problem in the two dimensional space. Multivariate normally distributed observations have been drawn randomly with a correlation of 0.3. The dashed lines show Ψ_x for two observations, $\boldsymbol{x_1} = (-1.5, NA)$, and $\boldsymbol{x_2} = (NA, -0.5)$. In this case the first principal component alone cannot explain the majority of variance, so we would assume $k = 2$. In this special case $\Omega = \Psi_x$, so the dashed lines represent the intersection of Φ and Ψ_x. Since the two projection methods above force us to choose k, so that $k + \tilde{m} \leq p$, we have to reduce k to one, when applying the exact projection algorithm. This is a serious problem, because this implies loss of information since components may be dropped randomly. Of course PCA returns an exact ranking of principal components, according to their size, but when lengths do not differ significantly, we cannot argue why we drop component a, and use component b, when a and b are similar in length.

When applying the exact projection method onto the system shown in Figure 4.4, we would have to drop one principal component, which here is obviously the second. However in high dimensional cases this decision is more complicated since we have to compare more than two components, several of them similar in length. When projecting the observations, displayed

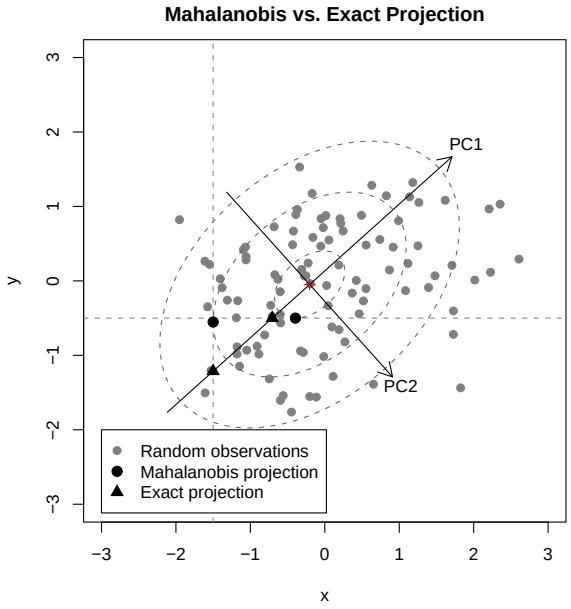


Figure 4.4: A random data set drawn from a multivariate normal distribution. Two observations are imputed, each by minimizing the Mahalanobis distance (dots) and by projection onto the first principal component (triangles).

as dashed lines, onto the first principal component we obtain estimations of the incomplete observations, which are appropriate, but we did not use all the information we have on our data structure. When projecting these observations in a way that the Mahalanobis distance is being minimized, we can use information of both principal components, so we are not confronted with the question which components to drop. The difference between the exact and Mahalanobis projection in Figure 4.4 is not negligible, so we have to make a decision for one of those methods.

As we have not yet developed any quality criterion, we can only argue formally: The lower the Mahlanobis distance of an observation, the higher is the probability that an observation occurs at that point, which is an argument for this approach. Dashed ellipses which represent all points with the Mahalanobis distance as the imputed observations, emphasize the difference of these two approaches, since an observation imputed by the exact projection algorithm can never be located inside such an ellipse, so its Mahalanobis distance will always be greater or equal to the distance of an observation imputed with the Mahalanobis method. Further the mentioned loss of information, caused by dropping principal components, is another argument against the exact projection method in such situations.

Since it is rather complex to express Ω formally, we will develop an approximative method, which delivers similar results: The idea behind this approach is to locate an observation in Ψ_x, so that its Mahalanobis distance in the p-dimensional space is being minimized. This is not exactly the same solution as searching a point $\in \Omega$ and minimizing the Mahalanobis distance in Φ, but this approach is nearly as good, awnd the development is not that com-

plex. When considering the formal definition of the squared Mahalanobis distance, for an observation $\boldsymbol{x}$,

$$\mathrm{MD}^2(\boldsymbol{x}) = (\boldsymbol{x} - \boldsymbol{T})^\top \boldsymbol{C}^{-1} (\boldsymbol{x} - \boldsymbol{T}) \tag{4.13}$$

with $\boldsymbol{T}$, the location and $\boldsymbol{C}$ the scatter estimate. However, we will make slight changes, in order to simplify the calculation. Since we know, that our data matrix has already been centered, we can omit $\boldsymbol{T}$. Further it is necessary to invert the covariance matrix $\boldsymbol{C}$, which can be very easily done, when this matrix is diagonal. This is the case when considering the data in the system of the principal components, where the covariance matrix

$$\boldsymbol{C}^{(pc)} = \boldsymbol{\Lambda} \tag{4.14}$$

can be expressed very simply. $\boldsymbol{\Lambda}$ is taken from Equation (2.2) (A diagonal matrix containing the eigenvalues of $\boldsymbol{C}$). Since we already have estimated the principal components, $\boldsymbol{\Lambda}$ is known, and so calculating $\left(\boldsymbol{C}^{(pc)}\right)^{-1}$ is trivial. When transforming the covariance structure in the system of principal components, we also have to express $\boldsymbol{x}$ transformed. We can do this, by using the expression for $\Psi_x^{(pc)}$, which we developed in the previous Section (Equation (4.7)). With all these changes the expression we have to minimize is

$$\mathrm{MD}^2(\boldsymbol{x}) = \left(\Psi_x^{(pc)}(\boldsymbol{\eta})\right)^\top \boldsymbol{C}^{-1} \left(\Psi_x^{(pc)}(\boldsymbol{\eta})\right) \tag{4.15}$$

$$\hat{\boldsymbol{\eta}} = \underset{\eta}{\operatorname{argmin}} \left(\mathrm{MD}^2(\boldsymbol{x})\right) \tag{4.16}$$

In this context $\boldsymbol{\eta} = \boldsymbol{x}_{\tilde{M}}$ again represents the missing values which influence the Mahalanobis distance of $\boldsymbol{x}$.

$$\begin{aligned}
\mathrm{MD}^2(\boldsymbol{x}) &= \left(\boldsymbol{\Gamma}_M^\top \boldsymbol{x}_M + \boldsymbol{\Gamma}_{\tilde{M}}^\top \boldsymbol{\eta}\right)^\top \boldsymbol{C}^{-1} \left(\boldsymbol{\Gamma}_M^\top \boldsymbol{x}_M + \boldsymbol{\Gamma}_{\tilde{M}}^\top \boldsymbol{\eta}\right) && (4.17)\\
&= \left(\boldsymbol{x}_M^\top \boldsymbol{\Gamma}_M + \boldsymbol{\eta}^\top \boldsymbol{\Gamma}_{\tilde{M}}\right) \boldsymbol{C}^{-1} \left(\boldsymbol{\Gamma}_M^\top \boldsymbol{x}_M + \boldsymbol{\Gamma}_{\tilde{M}}^\top \boldsymbol{\eta}\right)\\
= \boldsymbol{x}_M^\top \boldsymbol{\Gamma}_M \boldsymbol{C}^{-1} \boldsymbol{\Gamma}_M^\top \boldsymbol{x}_M &+ 2\boldsymbol{\eta}^\top \boldsymbol{\Gamma}_{\tilde{M}} \boldsymbol{C}^{-1} \boldsymbol{\Gamma}_M^\top \boldsymbol{x}_M + \boldsymbol{\eta}^\top \boldsymbol{\Gamma}_{\tilde{M}} \boldsymbol{C}^{-1} \boldsymbol{\Gamma}_{\tilde{M}}^\top \boldsymbol{\eta}\\
\frac{\partial}{\partial \boldsymbol{\eta}} &= 2\boldsymbol{\Gamma}_{\tilde{M}} \boldsymbol{C}^{-1} \boldsymbol{\Gamma}_M^\top \boldsymbol{x}_M + 2\boldsymbol{\Gamma}_{\tilde{M}} \boldsymbol{C}^{-1} \boldsymbol{\Gamma}_{\tilde{M}}^\top \boldsymbol{\eta} := 0 && (4.18)\\
\boldsymbol{\Gamma}_{\tilde{M}} \boldsymbol{C}^{-1} \boldsymbol{\Gamma}_{\tilde{M}}^\top \boldsymbol{\eta} &= -\boldsymbol{\Gamma}_{\tilde{M}} \boldsymbol{C}^{-1} \boldsymbol{\Gamma}_M^\top \boldsymbol{x}_M\\
\hat{\boldsymbol{\eta}} = -\left(\boldsymbol{\Gamma}_{\tilde{M}} \boldsymbol{C}^{-1} \boldsymbol{\Gamma}_{\tilde{M}}^\top\right)^{-1} &\boldsymbol{\Gamma}_{\tilde{M}} \boldsymbol{C}^{-1} \boldsymbol{\Gamma}_M^\top \boldsymbol{x}_M = \boldsymbol{A}\boldsymbol{x}_M && (4.19)
\end{aligned}$$

Again we obtained a simple transformation matrix $\boldsymbol{A}$, which computes the missing values of an observation so, that the Mahalanobis distance is as small as possible. An advantage of this transformation is, that $\boldsymbol{A}$ is independent of k, so we do not need to evaluate the number of relevant principal components in advance.

Since this projection matrix is calculated based on all principal components $k = p \Rightarrow k + \tilde{m} > p$, this projection can be done with any NA-pattern, which is not the case with the projections considered above. So the cross validation step for estimating k, as proposed by Walczak et al. (2001), described in Section (3.4.1) is unnecessary when applying this type of projection.

As discussed in 2.3.1, the estimation of k is a task which cannot really be formalized, so this characteristic of Mahalanobis distance based projection methods is really convenient.

As already mentioned, this method is only an approximation of the exact solution, which minimizes the Mahalanobis distance on hyperplane Ω generated by intersecting Φ and Ψ_x. Our imputed observation is not exactly located on Φ, but $\text{diag}\left(\boldsymbol{C}^{(pc)}\right)^{-1}$ can be considered as vector which penalizes deviations from the center with respect to the inverse length of principal components. So a deviation from the center of the structure in direction of a principal component $i > k$ which eigenvalue is very small will be penalized strongly, whereas a deviation in the direction of principal component 1, or $i \leq k$, which eigenvalue is rather large, will barely be penalized. So an observation imputed by this method will be forced into the structure of the data as good as possible, but when too few degrees of freedom are available for sufficiently moving an observation, it may not be exactly located on Φ in order not to change observed values.

When considering projection matrix $\boldsymbol{A}$ from Equation (4.19) and from Equation (4.10) it turns out, that the solutions are again similar, but this Mahalanobis distance based method uses all available information which has been obtained by the preceding PCA (eigenvectors *and* eigenvalues). The length of principal components, which are contained by $\boldsymbol{C}^{(pc)}$ is not considered in exact and over determined situations. Also Walczak et al. (2001) and Serneels et al. (2007) do not consider this information in their algorithm. So optimistically speaking, we can say, that since this third method can handle more information of our data structure, its results will be at least as good as the results of the algorithm proposed by Walczak et al. (2001) and Serneels et al. (2007).

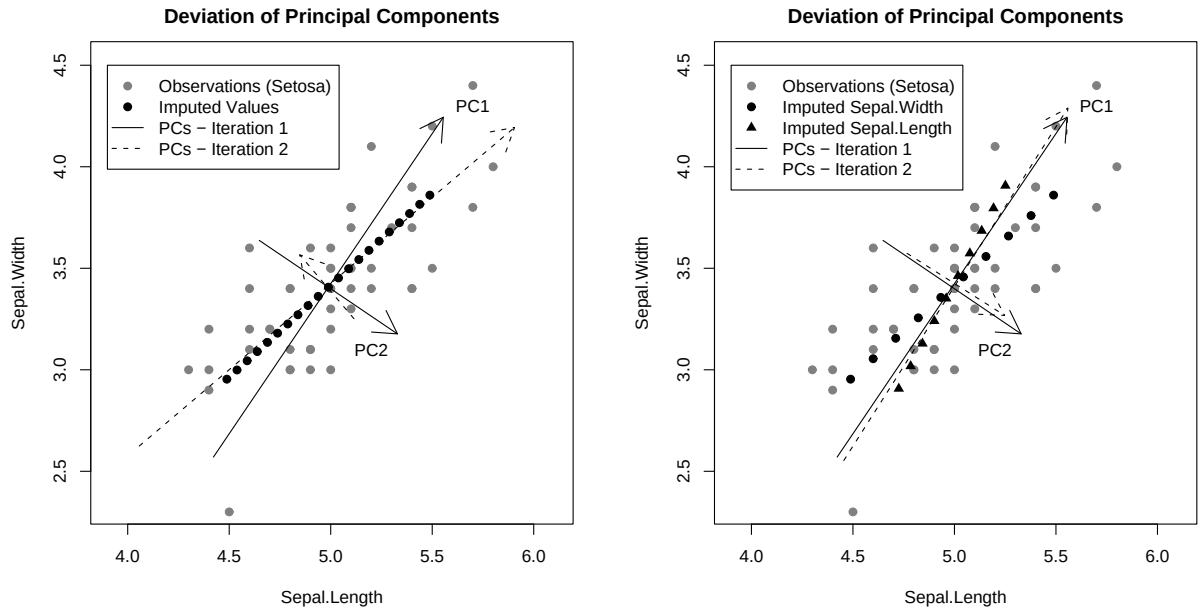
4.2.4 Drawbacks

As we do not project observations exactly onto Φ, in certain situations this behaviour may influence the estimation of principal components. This happens, when the amount of missings in one variable is much higher than in other variables.

Figure 4.5 (a) illustrates this issue by adding 20 observations to the iris data set, which all have a missing value in variable *Sepal.Width*. These observations have been imputed with the Mahalanobis method (k = 2). All of them attract the principal components in the same direction, which here is clockwise. So in the following iteration, estimated principal components are deviated noticeable.

In Figure 4.5 (b) missing values are distributed equally on both variables. The deviation of principal components in the second iteration is much smaller, since imputed observations are located on both sides of the *real* principal components.

A further problem, which points out in this context is, that imputation not only influences the direction of principal components, but also their length, the eigenvalues of the resulting structure. This is very conspicuous in Figure 4.5 (a), where the second component shrinks



(a) (b)

Figure 4.5: Fisher's iris data set - only showing observations of type *setosa*. The influence of non-uniformly distributed missing values (a) vs. uniformly distributed missings (b) on iterative PCA.

to half of its length during one iteration. The reason for this behaviour is, that observations are located around Φ, so the deviation from Φ is tried to be minimized by all of the discussed methods. A scale estimation in a direction which is orthogonal to Φ (e.g. PC2 in Figure 4.5 (a)) then may break down which is the reason for the shrinkage of principal components.

Both problems can also be noticed when applying the algorithm of Walczak et al. (2001) and Serneels et al. (2007) where iterative projection may even increase the deviation of principal components, so this is not a drawback of the Mahalanobis approach in particular.

4.3 Convergence Criterion

In step 4 of the algorithm proposed by Walczak et al. (2001), described in Section (3.4.1), convergence is mentioned without any suggestion how to obtain a convergence measurement. As Walczak et al. (2001) append Mathlab - code to their paper, it shows that they measure the absolute change of the estimated values. Serneels et al. (2007) proposes only to consider observations with a robust Mahalanobis distance below $\chi^2_{p,0.99}$, since strongly outlying observations may break down the convergence criterion.

However a big difference between the method of Walczak et al. (2001) and Serneels et al. (2007) and the direct projection method is, that in our case the location of imputed values only changes, when the directions of principal components change. This holds not true for

iterative projection as illustrated in Figure 3.4, where an observation is iteratively approaching towards a principal component, which does not change in that example.

This characteristic of direct projection enables us to develop a more straightforward convergence criterion. Since a movement of an observation can only occur, when the principal components move, we can measure convergence by calculating the difference of principal components within two iterations. Hubert et al. (2005) refer to Krzanowski (1979) who describes how to measure the angle between hyperplanes, spanned by two sets of principal components, here denoted by $\mathbf{\Gamma}$ and $\mathbf{\Pi}$. The angle between the systems of principal components is called *maxsub* in this context:

$$maxsub := \arccos\left(\sqrt{\lambda_k}\right) \tag{4.20}$$

whereas λ_k is the smallest eigenvalue of $\mathbf{\Gamma}_{.K}^{\top}\mathbf{\Pi}_{.K}\mathbf{\Pi}_{.K}^{\top}\mathbf{\Gamma}_{.K}$. When the systems of principal components are exactly orthogonal to each other, $maxsub = \frac{\pi}{2}$, so this measurement may be standardized by dividing it by $\frac{\pi}{2}$.

This method can be imagined as rotating p points, which all are on the axes of the original system of coordinates, into the system of the k principal components. Then these points are rotated back, but with the second rotation matrix. If the matrices are not equal, the points are not rotated back exactly onto the axes of the original system of coordinates, and so the deviation from those axes can be used to calculate the angle between the two sets of the principal components.

The advantage of using *maxsub* as convergence criterion is, that it does not depend directly on the imputed values, but on the estimation of the principal components. So this convergence criterion is automatically as robust as the applied PCA method itself, and thus cannot be affected by single outlying observations. So the decision which observations to consider for measuring convergence is dispensable, which is another advantage, since the $\chi^2_{p,0.99}$ - boundary cannot be argued formally (a 0.975 or 0.95 - quantile may also be appropriate in certain situations).

4.4 Implementation

When considering all aspects, which have been treated in the previous sections, we now can adopt the algorithm of Walczak et al. (2001) and Serneels et al. (2007):

1. If not already done, the data matrix is scaled and centered. Location and scale estimates are calculated columnwise, whereas missing values are omitted.

2. Missing values are initialized. When considering a location estimate as initial value, all missings may be set to zero, since the data is centered. Donor imputation, as described in Section 3.2 may also be considered here, in order to decrease the number of necessary iterations since the initial values are already plausible, and the change of these values

may not be that tremendous. However, usually initializing all missings with zero does it as well.

3. Create an index array (or sort the multivariate observations), such that observations with equal NA-patterns can be treated at once.

4. PCA is applied to the data set. Since we are interested in minimizing the influence of outliers on our estimation, a robust PCA-method is recommended.

5. Since PCA methods are usually based on their own center estimation, the data set has to be centered again, because the subsequent projection methods require exactly centered data.

6. One of the discussed projection methods is applied to each observation. As each of those methods depends on the current NA-pattern, a projection matrix must be calculated for each NA-pattern. At that point, the number of observed values m is relevant:

   - $m > k$: Apply the projection for over-determined systems (4.2.1).
   - $m = k$: Apply the projection for exact determined systems (4.2.2).
   - $m < k$: Apply the projection for under-determined systems (4.2.3).

   Observations with $m = 0$ or $\tilde{m} = 0$ are ignored.

7. Goto step 4 until convergence (until *maxsub* drops below a certain threshold), or a certain number of iterations is reached.

8. Undo centering from step 5. This can be done by adding all center vectors during the iteration, and subtracting this vector from the data matrix at the end.

9. Undo scaling and centering from step 1.

So we finally obtained an improved algorithm of the imputation method derived from the approach of Walczak et al. (2001), and robustified by Serneels et al. (2007). At this point we want to emphasize again, that Walczak et al. (2001) and Serneels et al. (2007) did not intend to develop an algorithm for imputation, but for (robust) PCA on data containing missings. Most of the issues and problems which have been discussed here, are only relevant when intending to impute values, and not when estimating principal components. As we will see shortly, the estimation of principal components cannot really be improved by this new algorithm, the only improvement affects the quality of imputation, which was not the aim of Walczak et al. (2001) and Serneels et al. (2007).

The source code of this algorithm has been appended to this thesis. It has been designed for the statistical environment R, with extensions written in C++. Since the proposed algorithm treats each NA-pattern individually, an implementation without the support of C++ would be computational inefficient. Further matrix multiplications with subsets of $\mathbf{\Gamma}$ like $\mathbf{\Gamma}_{MK}$ have been implemented very frugal. In environments like R such subsets of

matrices have to be copied into memory before being processed. The implementation in the appendix always operates on one and the same instance of $\mathbf{\Gamma}$, whereas subsets of matrices are virtually constructed by smart indexing. At that level matrices are no more represented by objects, but the memory where the single values are stored is accessed directly. So the mentioned operations which project observations into a subspace of the original structure can be implemented very efficient. Aside the improvement concerning runtime of the projection algorithm, which could be reduced by a factor far beyond 10, the memory consumption has been reduced enormously as well, compared to an according implementation in R.

4.5 Simulation Study

After theoretically discussing a new method for imputation of missing data, this algorithm shall be tested in order to assure, that it really performs better than existing approaches. For this purpose we will need to generate artificial data sets. The advantage of using artificial over real data sets is that we know the number of relevant components, the amount of white noise which has been added, and the *real* principal components.

4.5.1 Quality Criterion

In order to compare the quality of different imputation methods, we have to formulate an appropriate criterion. As we want to measure the precision of the imputation it is obvious to calculate the mean or median absolute difference between imputed and original values. Since we are not interested in the quality of the imputation of outlying observations, we will focus on the deviation of non-outlying observations only, and ignore the deviance of imputed outlying observations.

Further we apply PCA algorithms on imputed data, so we are interested in the quality of PCA estimation. On the one hand we want to know whether our projection method influences the estimation of principal components in a bad way, and on the other hand we want to know if we can keep up with the quality of the iterative projection method concerning the estimation of principal components. For this issue we will again consider the *maxsub* criterion as discussed in (4.3) which enables us to calculate the angle between *real* principal components, and the estimation returned by the imputation algorithm.

4.5.2 Generation of Test Data

For testing the described algorithms in an exactly known environment, we have to generate artificial test data with pre-defined characteristics, like a certain number k of relevant principal components. So we do actually not generate the data matrix $\boldsymbol{X}$ directly, but we generate the scores $\boldsymbol{Z}$, and transform them into the data matrix (Equation (2.3)). So the generation of the test data consists of generating the scores $\boldsymbol{Z}$, the loadings $\mathbf{\Gamma}$ and the eigenvalues, or lengths of the components $\mathbf{\Lambda}$.

1. A $n \times p$ matrix $\boldsymbol{Z^I}$ is created , with the values of the first k columns drawn from a standard normal distribution $N(0,1)$, and the columns $k+1,\ldots,p$ are set to zero. This matrix can be considered as *scaled scores*.

2. The p eigenvalues are defined by the following expression:
$$\text{diag}(\boldsymbol{\Lambda}) = \left(1^{-c}, 2^{-c}, \ldots, k^{-c}, 0, \ldots, 0\right) \tag{4.21}$$
with a constant $c \in R^+$.

3. The values of a matrix $\boldsymbol{\Gamma}$ of dimension $p \times p$ are drawn from a standard normal distributed, with the restriction, $\boldsymbol{\Gamma\Gamma}^\top = \mathbf{I}^p$.

4. Since principal components $\boldsymbol{Z^I}$ are now all of same length, they are *stretched* according to their eigenvalues:
$$\boldsymbol{Z} = \boldsymbol{Z^I\Lambda}^{0.5} \tag{4.22}$$

5. The scores are projected into the original p dimensional system of coordinates, which creates the data matrix:
$$\boldsymbol{X} = \boldsymbol{Z\Gamma}^\top \tag{4.23}$$

6. An error term $\boldsymbol{\varepsilon} \sim N_p(\mathbf{0}, \sigma_\varepsilon^2 \mathbf{I}^p)$ is added to each observation of $\boldsymbol{X}$, with σ_ε chosen with respect to the scale of the scores. An error term with larger variance than the underlying data makes reasonable imputation impossible. Since all eigenvalues of the structure are ≤ 1, σ_ε should not exceed a value of ~ 0.2 in order not to destroy the structure of the data completely.

7. A certain portion *p.blank* of data is randomly blanked, with equal probability for each cell.

Outliers

Since we also want to test the influence of outliers on our method, we also have to add some contamination to our data. This contamination is produced by drawing v observations from a different distribution, whereas two different types of outliers may be generated:

1. The center of the scores is moved by t^0:
$$\boldsymbol{Z}_j^* = \boldsymbol{Z}_j + \mathbf{1}_p t^0 \qquad , j = 1, \ldots, v, \tag{4.24}$$
whereas t^0 represents the shift of the outliers, and can be chosen arbitrarily. The smaller t^0 is chosen, the more similar are the distributions of *normal* and outlying observations.

2. The order of the scores of v observations is changed:
$$\boldsymbol{Z}_j^* = (\boldsymbol{Z}_{jp}, \ldots, \boldsymbol{Z}_{j1})^\top \qquad , j = 1, \ldots, v \tag{4.25}$$
Outliers generated this way are orthogonal to *normal* observations.

The portion of contaminated data will be referred to as *p.out*, whereas the type of outliers will be denoted as *type.out*.

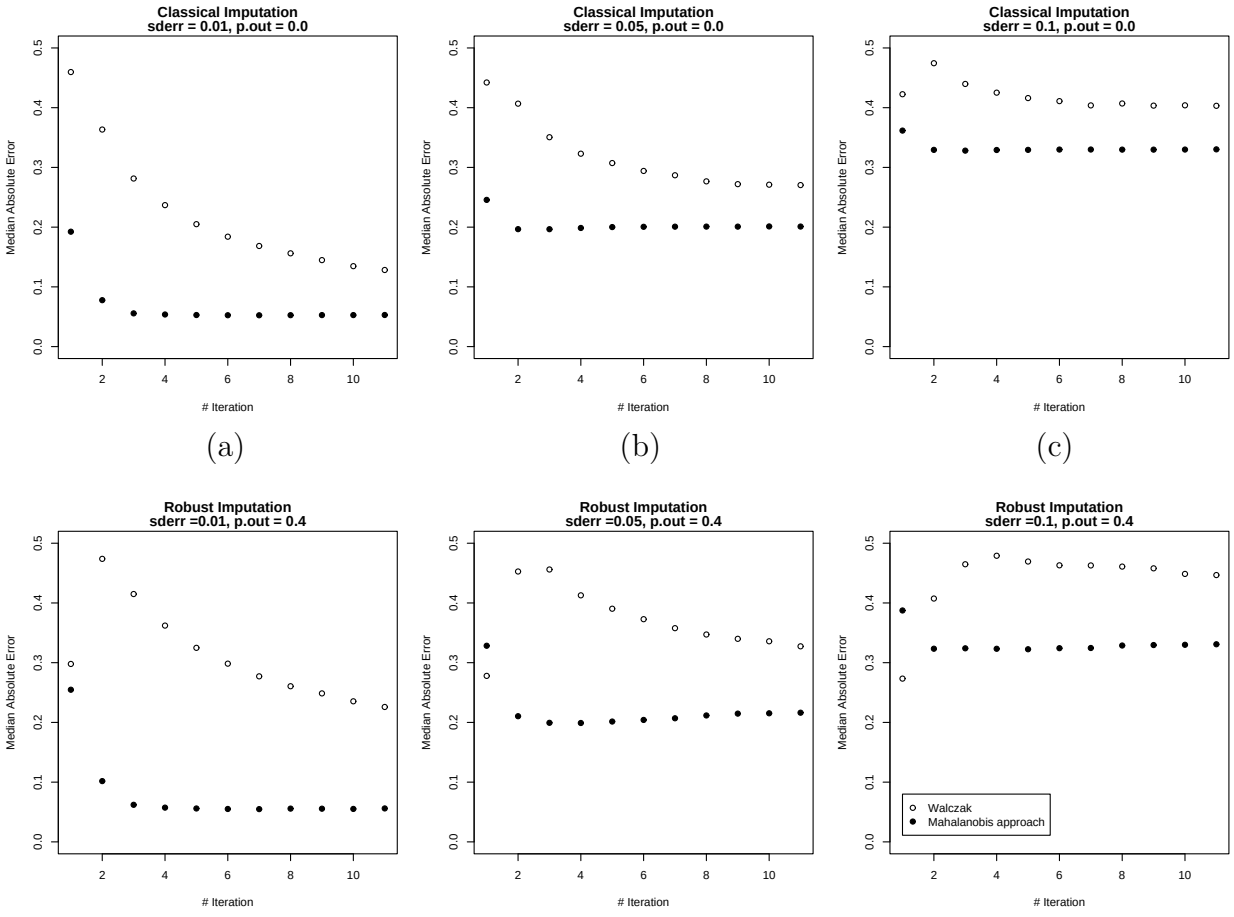


Figure 4.6: Median absolute errors returned by the discussed imputation methods for $\sigma_\varepsilon = 0.01$, 0.05 and 0.1

4.5.3 Iterative Behaviour

Since we have made severe changes which affect the iterative projection algorithm of Walczak et al. (2001), we will consider the performance of both algorithms during execution. So we will calculate the median absolute error of the imputed values and the deviance of principal components in each iteration. For this purpose a dataset with the following parameters will be drawn randomly, and the appearing error will be logged: ($n = 1000$, $p = 10$, $k = 6$, $c = 1.5$, $p.out = 0.4$, $p.blank = 0.2$).

Figure 4.6 shows the median absolute errors of the imputed values after applying both imputation algorithms. Black dots represent the Mahalanobis distance approach, white dots represent the error of the approach of Walczak et al. (2001) and Serneels et al. (2007). The classical method has been applied on a dataset without outliers, and the robust method with MCD as covariance estimator has been applied on a dataset with 40% outliers of type 1, as described in Equation (4.24), with $t^O = 5$. The process of imputing missing values of a

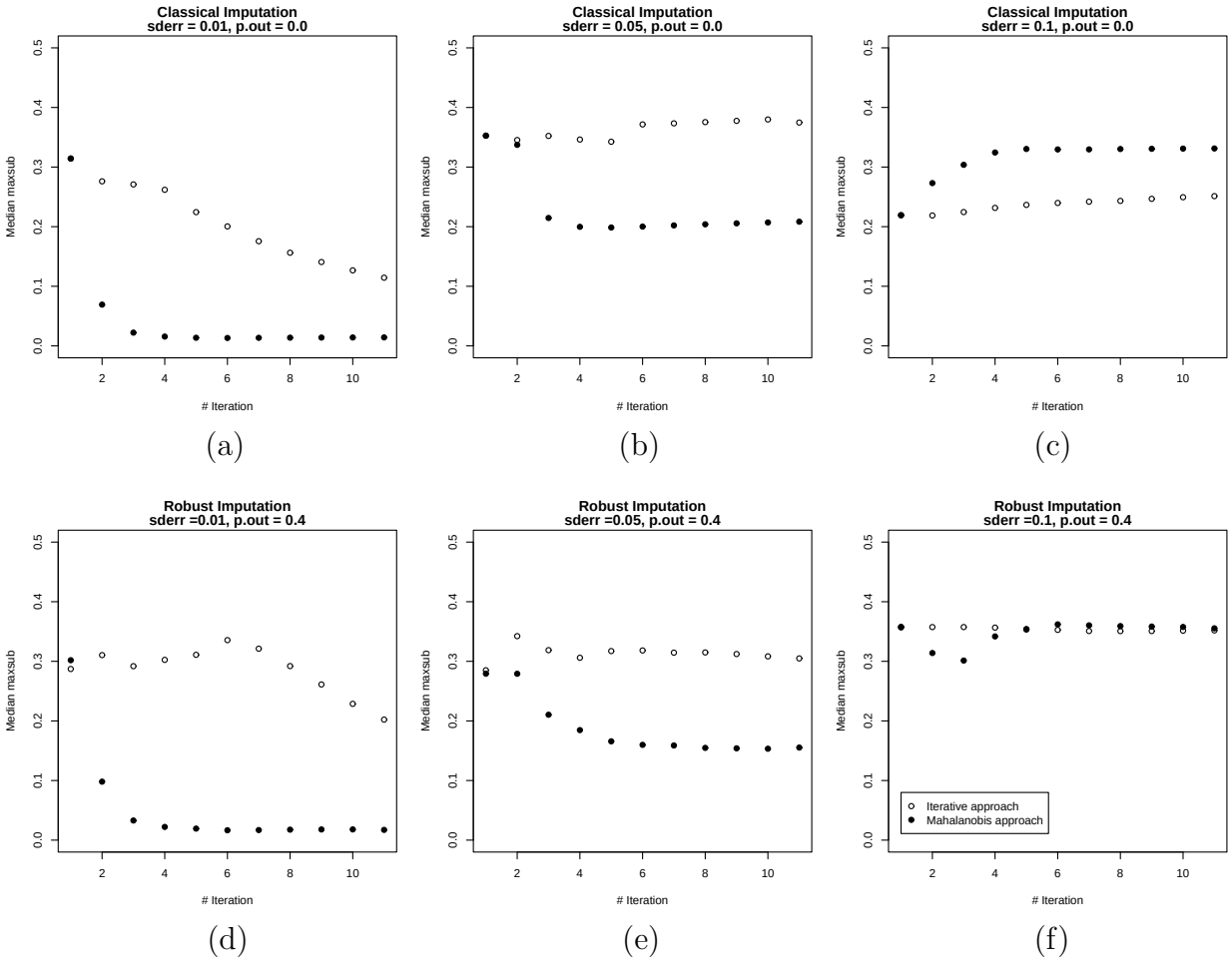
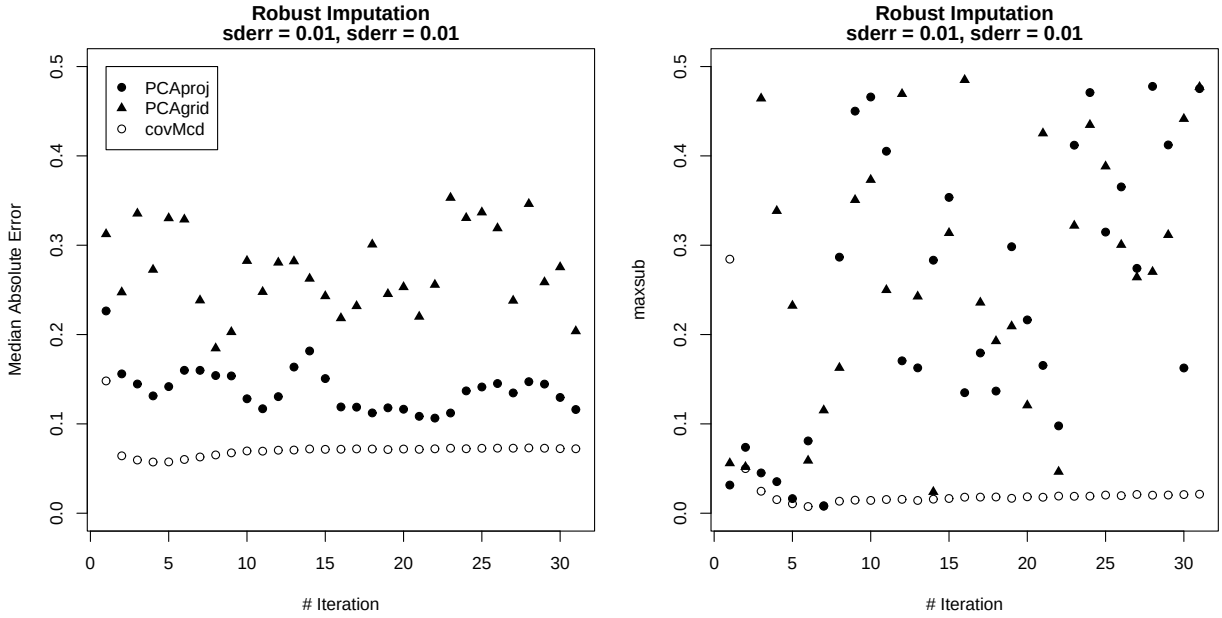


Figure 4.7: Median *maxsub* criterion returned by the discussed imputation methods for σ_ε = 0.01, 0.05 and 0.1

blanked data matrix has been repeated 20 times for each parameter constellation and the median of the errors has been calculated in order to obtain a representative estimation of the error.

The plots show how the median absolute error of imputed values changes, while repeatedly estimating principal components, and applying the mentioned projection methods. The error of the iterative algorithm changes significantly from iteration to iteration, whereas the error of the Mahalanobis distance approach is rather constant, except for the first iteration. So terminating the Mahalanobis distance based algorithm after two iterations seems to be as good, as waiting for the algorithm to converge. The approach of Walczak et al. (2001) needs more iterations until convergence, and there are even situations, where the error increases significantly , so the principal component estimation seems to be influenced badly by imputed values. In the following simulation we will compare the results of the converged approach of Walczak et al. (2001) and the Mahalanobis distance approach terminated after two iterations in order to get an idea whether the early terminated algorithm can keep up



(a) (b)

Figure 4.8: Performance of different PCA methods in combination with the Mahalanobis distance based projection approach. Quality of imputation (a) and PCA (b).

with the converged iterative algorithm. In these plots the Mahalanobis distance based algorithm performs persistently better than the iterative approach.

Figure 4.7 shows the deviance of principal components, measured with the *maxsub* criterion. The lower these values are, the better is the estimation of principal components. Considering this criterion, the algorithm needs more iterations in order to obtain a stable level, which is contradictious to the assumption that the result of the Mahalanobis approach does not improve after two iterations. This may be caused by the error term ε added to the data, which reduces the correlation between the quality of principal component- and missing value estimation. This is very outstanding in situation (c), where PCA performs better with the iterative approach, but the performance regarding imputation is still better with the Mahalanobis method.

4.5.4 Alternative PCA Methods

One further aspect which should be analyzed is the behaviour of the $PCAproj$ (Croux et al., 2005) and $PCAgrid$ (Croux et al., 2007) algorithm in connection with the developed projection method. The simulation setup is similar to the previous situation, but we will only add an error $\boldsymbol{\varepsilon} \sim N_p\left(\mathbf{0}, 0.01^2\mathbf{I}^p\right)$ and we do only consider the result of one single run and not calculate the median absolute errors of several runs. The PCA methods $PCAproj$, $PCAgrid$ from the R-package *pcaPP*, and PCA based on covariance estimation with MCD have been used. The results can be seen in Figure 4.8. In plot (a) the error of imputed values is shown,

which are highly auto-correlated when using MCD for PCA. When applying $PCAproj$ or $PCAgrid$ these values are scattered strongly, so the iterative improvement of the imputed values fails completely. This effect is even stronger, when considering the $maxsub$ criterion which is very low and stable when applying the MCD estimator, so principal components are estimated pretty well with this method. Principal components estimation with the $pcaPP$ functions again fail completely, whereas the quality of the estimation is no more auto-correlated at all, and seems to be random. So in the context of the Mahalanobis distance projection algorithm, PCA based on MCD covariance estimation can be recommended, whereas $pcaPP$-methods do not work at all. This may be due to the special characteristics of these algorithms, which are incompatible with the considered projection methods.

4.5.5 Performance Tests

For testing the performance of the developed algorithm, the median absolute imputation error of several simulations with different parameters has been acquired. Tests with different types and portions of outliers have been simulated. σ_ε has been set to 0.01 and the portion of outliers has been raised from 0% up to 30%. The amount of missing values over the complete dataset was fixed to 20%. For generating outliers of type 1, t^0 has been set to 5. Again the test-data comes from a 10 dimensional multivariate normal distribution, with 6 relevant principal components. Each test has been repeated twenty times with the same parameter set, and the median of the resulting errors has been determined. The number of principal components which should be considered for imputation has been iterated from 1 up to 10, so each possibility in this context has been tested. Several imputation methods have been applied: On the one hand the iterative algorithm, as well as the approach developed here with only two iterations. On the other hand a kNN algorithm as proposed by Hastie et al. (1999) has been applied too, in order to get an idea how the algorithms perform in comparison to common imputation methods. Further principal components have been estimated regarding the proposals of Walczak et al. (2001) and Serneels et al. (2007), and afterwards the Mahalanobis projection method has been applied with $k = p$, which is in the plot referred to as the combination of the iterative and Mahalanobis approach.

Figure 4.9 shows the corresponding results. In general the projection algorithm developed here, in combination with the iterative estimation of missing values performs better than the imputation method based on Walczaks proposal. In order to compare the performance of the Mahalanobis based approach with $k = p$ with the other results, a corresponding horizontal dashed line has been added, which represents the error the Mahalanobis approach produces when setting $k = p$. This line is in most cases lower than each result of projection based algorithms, so when applying this method with simply setting $k = p$ seems to be an appropriate approach which can be calculated as quick solution. Improvements of this first result by modifying k can be made afterwards, but are computational very expensive, since the *real* k is unknown.

Since there are several parameters which can be modified when defining such simulations ($p.out$, $p.miss$, σ_ε, p, k, c), these 10 plots do only represent a subset of possible situations

which can be simulated. There might be situations where even the iterative algorithm performs better. Especially when a very large error term is added onto the dataset the difference in quality between both algorithms decreases drastically. However in common situations the advantage of the developed algorithm can be shown with the plots in Figure 4.9. The advantage which points out by the these plots are the runtime, which can be reduced significantly, since we can usually terminate the Mahalanobis distance based algorithm after two iterations, and on the other hand, that the estimation of k can be omitted.

4.6 Conclusion and Future Development

Up to this point a new robust imputation algorithm, based on a proposal of Walczak et al. (2001) and Serneels et al. (2007) has been developed, which estimates missing values according to principal component estimates. As simulation shows, this new algorithm performs better than the approach of Walczak in most cases, however when the amount of white noise increases, a kNN approach still delivers even better results. Since the behaviour of the approach presented here has not been analyzed completely, there still might be aspects which have not been considered yet, and which may help to improve the quality of this method in future research.

Problems which will be considered next would be the application of this method on data sets containing nominal and categorical variables as well as extending this approach for multiple imputation. Since the Mahalanobis distance approach presented here is only an approximation, it may be improved by exactly defining Ω, the hyperplane of intersection, and then minimizing the Mahalanobis distance in this subspace. This has not been done yet due to the mathematical complexity, which goes beyond the scope of this thesis. Such developments should increase the quality of the algorithm, and maybe it will be possible to obtain even better results than the kNN-algorithm delivers, in all situations.

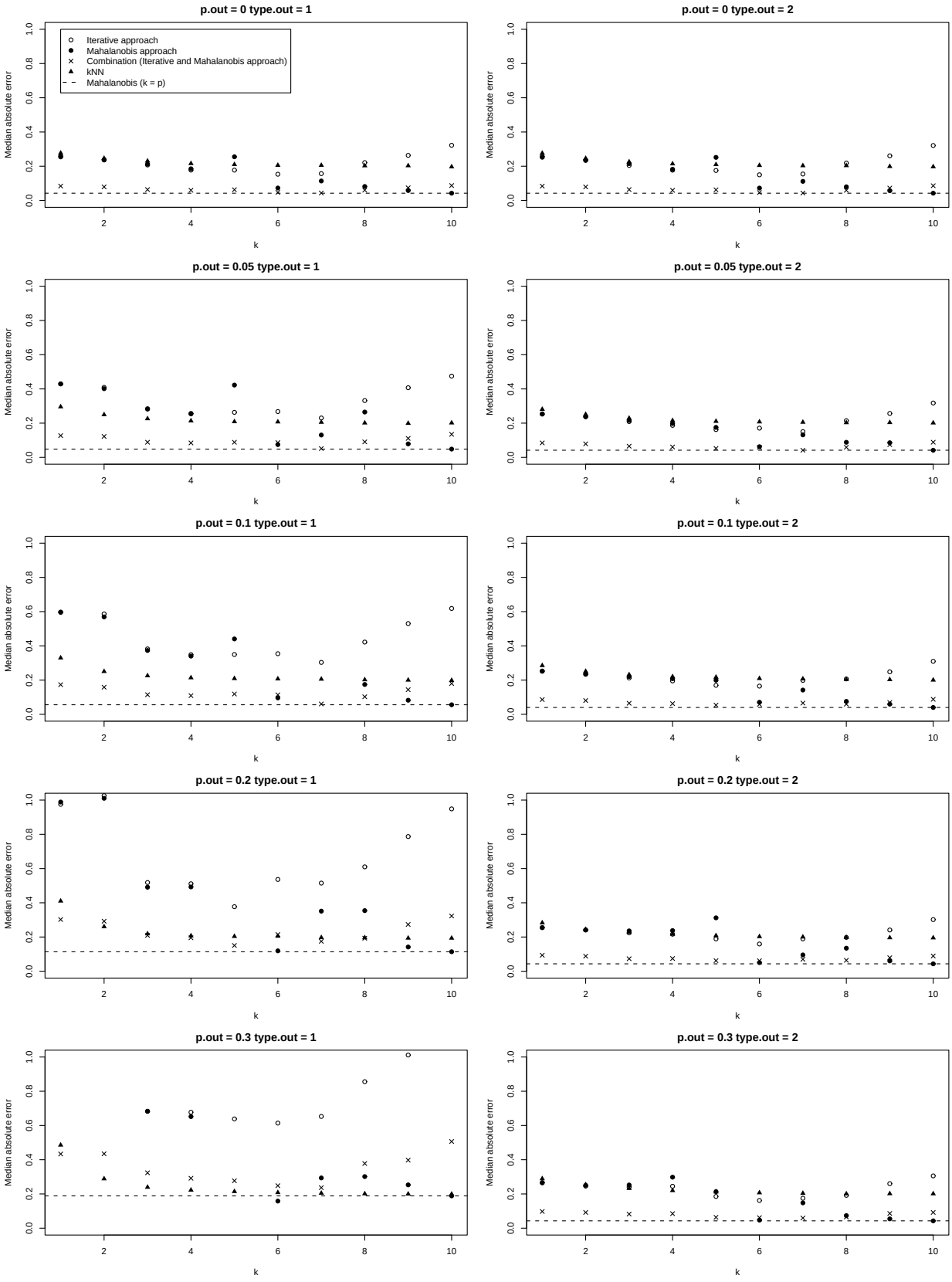


Figure 4.9: Median absolute errors of different imputation algorithms.

Appendix A

Source Code (R)

```
gentest <- function (n = 1000, p = 10, k = 5, sderr = 0.01, exp = 1.5,
                     p.out = 0, loc.out = 5, type.out = 1)
{
##########################################################################
##
## AIM:   Generate random testdata
##
## INPUT: n        number of observations to generate
##        p        number of variables per observation
##        k        number of relevant principal components in the dataset
##        sderr    standard deviation of error term
##        exp      exponent for calculating the deviation of scores
##        p.out    portion of outliers
##        loc.out  location of outliers
##        type.out type of outliers
##
## OUTPUT:   x     n x p data matrix containing the testdata
##         cov     theoretical covariance structure of data
##         center  center of the data (= 0) -  just for completeness
##         sdev    standard deviation of scores
##         n.out   number of outliers generated
##         obsok   logical array indicating outliers (FALSE)
##                    and regular observations (TRUE)
##
##########################################################################

      ##  generate scores
      sdev =  ((1:k)^-exp)
  td <- matrix (rnorm (n * k, rep (0, k),sd = 1),
                ncol = k) %*% diag (sdev)
```

```
  td = cbind (td, matrix (0, nrow = n, ncol = p-k))

  p.out = min (max (p.out, 0), 1)
  n.out = as.integer (n * p.out)
  fn = n - n.out
  obsok = rep (T, n)
  if (n.out > 0)
  {
    if(type.out == 1)
      td [fn:n, ] = td [fn:n, ] + loc.out    ## creating outliers
    else
      td [fn:n, 1:p] = td [fn:n, p:1]
    obsok [fn:n] = FALSE
  }

      ## randomly rotating data
  rot = qr.Q(qr(matrix(rnorm(p * p), p)))
  tdr <- td %*% t(rot[,1:p])

      ## adding error term
  tdr = tdr + rnorm (n * p, mean = 0, sderr)

      ##  calculating covariance structure of the data
#  datacov = t(rot) %*% diag(c((1:k)^-exp,rep (0, p-k))) %*% rot
  datacov = (rot) %*% diag(c(sdev^2,rep (0, p-k))) %*% t(rot)

      ##  adding error to theoretical covariance structure
  datacov = datacov + diag (rep (sderr^2, p))

  list (x = tdr, cov = datacov, center = rep (0, p),
          sdev = c(sdev^2 ,rep (0, p-k)), n.out = n.out,
          idx.out = fn:n, obsok = obsok, loadings = rot)
}
```

```
blank = function (x, p = 0.1)
{
##############################################################################
##
## AIM:   Blanks values of a data matrix
##
## INPUT:  x   the data matrix to be blanked
##         p   the portion of values to blank
##
## OUTPUT:     the blanked data matrix
##
##############################################################################

        np = length (x)

                        ##        draw random indices which will be blanked
        blank.idx = sample (np, as.integer (np * p))
        ret = as.numeric (x)

                        ##        blank data
        ret [blank.idx] = NA
        ret = matrix (ret, ncol = ncol (x))

                        ##        copy attributes of input matrix
        attributes (ret) = attributes (x)
        ret
}
```

```
pcImpute <- function (x, method="robust", eps=0.1, k,maxiter = 30,
                      convmode = 1, projfunc = pcNAProj, PC.control,
                      scale = T, kreal, ...)
{
##########################################################################
##
## AIM:   Imputes missing values according to principal components
##
## INPUT: x         The data matrix containing observations with missings
##        method      Specifies the PCA - method. allowed values are:
##               "robust"   PCA, based on covMCD
##               "PCAgrid"   Perform the grid-algorithm from Package pcaPP
##               "PCAproj"   Perform the proj-algorithm from Package pcaPP
##               else        do classical PCA
##        eps        threshold for convergence
##        k          number of relevant principal components
##        maxiter    maximum number of iterations
##        projfunc   which function to use for projecting the data onto PCs
##        PC.control a list containing parameters for the PCA method
##        scale      Logical value indicating whether the data matrix
##                     should be scaled previously
##
## OUTPUT: x         a data matrix containing the complete dataset
##         niter     the number of executed iterations
##         converged a logical value indicating whether the algorithm has
##                     converged
##         pc        the last estimation of PCs
##         gcm       the center which has been used for centering in advance
##         gsd       the scale which has been used for scaling in advance
##         angleLD   if realPC has been specified, this vector returns the
##                     angle between estimated and original PCs
##         err       if realDat has been specified, this vector returns the
##                     median absolute error between estimated and original
##                     data
##
##########################################################################

  if (sum (is.na(x)) == 0)
    return (0) ;

  xorig = x

  ##  convmode: 1 .. serneels, 2 .. Hubert
  ##  nSWMaxIter - when m > k --> serneels & wlaczak projection necessary
```

```
p= ncol (x)

if (missing (k))
  k = p - 1
if (k > p)
{
  k = p-1
  warning ("k reduced to p-1")
}
n = nrow (x)

    ##  calculate NA-pattern for all observation
I = !is.na (x) + 0

    ##  Generates an index vector which orders the observations
    ##  according to their NA-pattern
pat = CalcNAPattern_II (x, zeroisone = 0)

w <- which(is.na(x), arr.ind=TRUE)
w <- w[order(w[,1]),]      ##  sort w for pcImpDataDistance.C which
                           ##  requires a sorted index structure
NACount = nrow (w)

ws <-  apply (x, 1, function(x) {sum (is.na(x))})
wr <- ws != 0
sumws = sum(ws)

nm = sum (wr)                  ##  # observations with missings

gcsd = rep (1, p)

if(any (method == c("robust", "PCAgrid", "PCAproj")))
{    ##  calculate roust center and scale
  gcm <- apply (x, 2, median, na.rm = T)
  if (scale)
    gcsd <- apply  (x, 2, mad, na.rm = T)
}
else
{    ##  calculate classical center and scale
  gcm <- colMeans (x, na.rm = T)
  if (scale)
    gcsd <- sd(x, na.rm = T)
}
```

```
    ## z-transformation of the data matrix
x = scale.C (x, gcm, gcsd)    ##  Z - trans

    ## initializes missings with a center estimate (= 0)
x[w] =  0

d <- 1
niter = 0

D= array (NA, 2)
convmode = min (2, max (1, convmode))

while(d > eps && niter <= maxiter)
{    ##  do until convergence or maxiter has been reached
  xneu = x

    ##  the specified PCa method is applied on the current data matrix
  if( method == "robust" )  ##  calculate robust ...
  {
    xneu[w] = xneu[w] + rnorm (NACount, sd = 0.005)
    if (missing (PC.control))
      pc = princomp (cov = covMcd(xneu))
    else
      pc = princomp (cov = covMcd(xneu, control = PC.control))
  }
  else if (method == "PCAgrid")
    pc = PCAgrid (xneu, k = p, control = PC.control)
  else if (method == "PCAproj")
    pc = PCAproj (xneu, k = p, control = PC.control)
  else            ##  ... or classical covmat
    pc = princomp  (xneu)

    ##  center current data matrix according to the PCs
  xneu = scale.C (xneu, center = pc$center)

    ##  apply projection function on the current dataset
  xneu = projfunc (x = xneu, pc = pc, k= k, NAPattern = pat[[1]],
         NAOrder = pat[[2]], I = I, ...)

  scores = xneu %*% pc$loadings  ##  calculate scores

    ##  undo centering
  xneu <- scale.C (xneu, center = pc$center, zTrans = F)
```

```
    {
      ##  calclate mahalanobis distances of imputed observation
      ##    (convergence should not be measured based on outliers..)
      obsok = mahalanobis (scores, center = rep (0, p),
                  cov = diag (pc$sdev^2))[wr] <  (qchisq (df = p, 0.99))

      ##  only observations which were in proper range
      ##    (mahabdist < chisq(df, .99)) shall be considered
      curobsok = obsok #& lastobsok

      if (any(curobsok))
        D[1] = pcImpDataDistance (x, xneu, w, curobsok)
      else
      {
        D[1] = 1
        print ("Switching to convmode 2!")
        convmode = 2
      }
    }

    D[2] = 1
    if (niter >= 1)
    {  ##  calculate angle between PCs of two iterations as convergence
       ##    criterion
      D[2] = CompPCs.R (oldLd, pc$loadings, kreal)
    }
    print (c(D[2], NA, D[1], mean (obsok)))
    d = D[convmode]
    oldLd = pc$loadings

    x[w] <- xneu[w]

    niter = niter + 1
  }
      ##  reverse Z - transformation from the beginning
  x = scale.C (x, gcm, gcsd, F)

  list (x = x, niter = niter, converged = niter < maxiter, pc = pc,
        gcm = gcm, gcsd = gcsd, angelLD = angelLD, err = err)
}
```

```
iterproj = function (x, pc, k, NAPattern, NAOrder, nSWMaxIter = 20, I)
{
###############################################################################
##
## AIM: wrapping function for "iterproj", implemented in C++
##      implementation of the projection algorithm by walczak
##      projects observations containing missings on a hyperplane
##        spanned by k PCs
##
## INPUT: x          a data matrix
##        pc         a structure containing the principal components
##        k          number of relevant components
##        NAPattern  an optional vector identifying the NA pattern of
##                   each observation
##        NAOrder    an optional index vector which orders observations
##                   according to their pattern
##        nSWMaxIter the maximal number of iterations of the algorithm
##        I          an optional matrix which identifies observed (1)
##                   and missing (FALSE) values
##
##   OUTPUT:         a matrix containing the projected observations
##
###############################################################################

     ##  if not specified: calculate I
  if (missing (I))
    I = !is.na(x) + 0

     ##  if not specified: calculate NA-pattern
  if (missing (NAPattern) || missing (NAOrder))
  {
    pat = CalcNAPattern_II (x, zeroisone = 0)
    NAPattern = pat$NAPat
    NAOrder = pat$NAOrd
  }

  matrix (
  .C ("iterproj",
    as.integer (c(nrow(x), ncol (x), k, nSWMaxIter)), x = as.double (x),
    as.double (pc$loadings), as.integer (NAPattern), as.integer (NAOrder),
    as.integer (I), packate = "fImp", NAOK = TRUE )$x, ncol = ncol (x))
}
```

```
pcMahaProj <- function (x, pc, k, NAPattern, NAOrder, boost = F, I)
{
##########################################################################
##
## AIM: wrapping function for "mahaproj", implemented in C++
##      implementation of the projection algorithm developed in
##        this thesis
##      projects observations containing missings on a hyperplane
##        spanned by k PCs
##
##  INPUT: x          a data matrix
##         pc         a structure containing the principal components
##         k          number of relevant components
##         NAPattern  an optional vector identifying the NA pattern of
##                      each observation
##         NAOrder    an optional index vector which orders observations
##                      according to their pattern
##         nSWMaxIter   the maximal number of iterations of the algorithm
##         I          an optional matrix which identifies observed (1)
##                      and missing (FALSE) values
##
##  OUTPUT:           a matrix containing the projected observations
##
##########################################################################

  if (!is.matrix (x))
  {
    x = as.matrix (x)
    if (ncol(x) == 1 && nrow (x) > 1)
      x = t(x)
  }
      ##  if not specified: calculate I
  if (missing (I))
    I = !is.na(x) + 0
      ##  if not specified: calculate NA-pattern
  if (missing (NAPattern) || missing (NAOrder))
  {
    pat = CalcNAPattern_II (x, zeroisone = 0)
    NAPattern = pat$NAPat
    NAOrder = pat$NAOrd
  }

  p = ncol (x)
  n = nrow (x)
```

```
  if (p != dim(pc$loadings)[1])
    stop (paste ("Loadings have wrong dimension (should be ", p, ")",
          sep =""))

  if (boost)
    #pc$sdev = pc$sdev - pc$sdev[p] + 0.0001
    pc$sdev[p] =0.0001

  if (missing (k))
    k = sum (pc$sdev > 0.0001)

  ret = .C("mahaproj", ret = as.integer (c(n,p,k)), x = as.double (x),
    as.double (pc$loadings), as.double (pc$sdev), as.integer (NAPattern),
    as.integer (NAOrder), as.integer (I), package = "fImp", NAOK = TRUE)
  matrix (ret$x, ncol = p)
}

scale.C <- function (x, center = rep (0, ncol(x)),
                     scale = rep (1, ncol(x)), zTrans = T)
{
##############################################################################
##
## AIM:   wrapper function for "scale" - written in C++
##        fast centering and scaling
##
## INPUT: x       data matrix
##        center  center applied to the data matrix (numerical vector)
##        scale   scale applied to the data matrix (numerical vector)
##        zTrans  logical value indicating whether regular zTrans (TRUE)
##                  or inverse zTrans (FALSE) should be done
##
## OUTPUT:        centered and scaled data matrix
##
##############################################################################

        matrix (
        .C("scale",
                ret = as.integer (c(nrow(x), ncol(x), zTrans)),
                x = as.double (x), as.double (center), as.double (scale),
                package = "fImp", NAOK = TRUE)$x, ncol = ncol(x))
}
```

```
CalcNAPattern_II <- function (x, zeroisone = 1)
{
##############################################################################
##
## AIM:    Wrapper for "CalcNAPattern_II" implemented in C++
##         The function sorts all observations according to their
##         NA-pattern and returns the corresponding index - array
##
## INPUT:  x         data matrix
##         zeroisone value for conversion between 0/1 indices
##           (first element is 0/1)
##
## OUTPUT: NAPat     a identifier for each obsevations NA-pattern
##         NAOrd     index vector which orders observations according
##                     to their NA-pattern
##
##############################################################################

        n = nrow (x)
        ret = .C("CalcNAPattern_II",
                as.integer (c(n,ncol(x))),
                as.double (x),
                NAPat = integer (n),
                NAOrd = integer (n),
                package = "fImp",
                 NAOK = TRUE
        )
        ret$NAOrd = ret$NAOrd + zeroisone
        return (ret[3:4])
}
```

```
iterproj.R = function (x, pc, k, NAPattern, NAOrder, nSWMaxIter, I)
{
##########################################################################
##
## AIM: Does Walczak - projection (one iteration)
##
##  INPUT: x          a data matrix
##         pc         a structure containing the principal components
##         k          number of relevant components
##         NAPattern  dummy - for compatibility with other methods
##         NAOrder    dummy - for compatibility with other methods
##         nSWMaxIter dummy - for compatibility with other methods
##         I          dummy - for compatibility with other methods
##
##########################################################################

        x %*% pc$loadings [,1:k] %*% t(pc$loadings [,1:k])
}

CompPCs.R <- function (ld1, ld2, k)
{
##########################################################################
##
## AIM:    comparison of two systems of principal compontnts given by
##         loadings ld1 and ld2 with k interesting components
##
## INPUT:  ld1, ld2  loadings of systems which shall be compared
##         k         number of relevant components
##
## OUTPUT: angel between ld1 and ld2
##
##########################################################################

        sld = t(ld1[,1:k]) %*% ld2 [,1:k] %*% t(ld2[,1:k]) %*% ld1[,1:k]

        esld = eigen(sld)
        acos (sqrt (esld$values [k])) * 2 / pi
}
```

Appendix B

Source Code (C++)

```
  class CNAPattern
  {
////////////////////////////////////////////////////////////////////////
//
//  AIM:  Node for the binary tree, created to sort the NA-patterns
//  For sorting the observations according to their NA-patterns a binary
//  tree is created. each level of the tree represents a variable. if an
//  observation has an missing value at the respective variable, the it is
//  stored in the "left" subtree, otherwise in the "right"
//
////////////////////////////////////////////////////////////////////////

  public:
    CNAPattern () ;
    ~CNAPattern () ;
    CNAPattern *m_pNA ;  // pointer to "left" item
                         // in the last level not other CNAPattern objects
                         //   are referred, but an index array of the
                         //   corresponding observations
    CNAPattern *m_pNum ; // pointer to "right" item
                         // in the last level not other CNAPattern objects
                         //   are referred, but the number of corresponding
                         //   observations
    char m_cLastLevel ;  // indicating whether this object belongs to the
                         //   last level of the tree
  } ;

  CNAPattern::CNAPattern ()
```

```
  {
    m_pNA = NULL ;      //  or int *
    m_pNum = NULL ;     //  or int
    m_cLastLevel = 0 ;
  }

  CNAPattern::~CNAPattern ()
  {
    if(m_cLastLevel)
    {
      delete [] (int *) m_pNA ;
      return ;
    }
    if (m_pNA)
      delete m_pNA ;
    if (m_pNum)
      delete m_pNum ;
  }

  void WriteNAPattern (CNAPattern *pNA, int *&pnPattern, int *&pnOrder)
  {
////////////////////////////////////////////////////////////////////////
//
//  AIM:  Recursively writes an index vector from nodes pNA to pnPattern
//          and stores the pattern IDs
//
//  INPUT: pNa        the parent (topmost) node of the binary tree
//         pnPattern  the array with pattern ids to be filled
//         pnOrder    the index array to be filled
//
////////////////////////////////////////////////////////////////////////

    if (pNA->m_cLastLevel)
    {  //  if already reached bottom level: fill pnPattern und pnOrder
      memcpy (pnOrder, pNA->m_pNA ,sizeof (int) * (int ) pNA->m_pNum) ;
       // copy index array of current NA-pattern to the big index-array
      pnOrder += (int ) pNA->m_pNum ;
      int i ;

      //  write a(any) pattern ID - since only the current position of the
      //  index array is available, which differs for each call of this
      //  function, the pointer to the current position in the index
      //  array is used as pattern ID
```

```
      for (i = (int ) pNA->m_pNum - 1; i>= 0; i--)
        *pnPattern++ = (int) pnOrder ;
      return ;
    }

    // do recursions for both branches of the current node
    if (pNA->m_pNA)
      WriteNAPattern (pNA->m_pNA, pnPattern, pnOrder) ;
    if (pNA->m_pNum)
      WriteNAPattern (pNA->m_pNum, pnPattern, pnOrder) ;
  }

  void CalcNAPattern_II (int *pnPar, double *pdDat, int *pnPattern, int *pnOrder)
  {
////////////////////////////////////////////////////////////////////////
//
//  AIM:  calculates an index array which orders the observations
//        according to their NA-pattern. Further an pattern ID is assigned
//        to each observations. This ID is equal, when two observations
//        have the same NA-pattern. If the NA-pattern differs, these IDs
//        differ too.
//
//  INPUT: pnPar      array containing parameters:
//            0:       rowNum of matrix pdDat
//            1:       colNum of matrix pdDat (not needed here)
//         pdDat       Data Matrix containing observations to project (nxp)
//         pnPattern   an array which is filled with the pattern IDs
//         pnOrder     the index array which is calculated
//
////////////////////////////////////////////////////////////////////////

    int &n = pnPar [0] ;
    int &p = pnPar [1] ;

    int i, j ;

    CNAPattern *pNAStart = new CNAPattern ;
    CNAPattern **ppCurNA = &pNAStart ;

    double *pdCurVal ;

    //  creating the tree:
    for (i = n - 1; i >= 0; i--)
    {  // iterate through all observations
```

```
  pdCurVal = pdDat + i ;

  ppCurNA = &pNAStart ;
  for (j = p - 1; j >= 0; j--)
  {  //  iterating through all variables & stepping down the tree
    if (R_IsNA (*pdCurVal))
      ppCurNA = &(*ppCurNA)->m_pNA ;
    else
      ppCurNA = &(*ppCurNA)->m_pNum ;

      // if the according node has not yet been created:
    if (!*ppCurNA)
      *ppCurNA = new CNAPattern ; //  create node
    pdCurVal += n ;
  }
    //  set last level-flag for last level-node
  (*ppCurNA)->m_cLastLevel = 1 ;
    // increases the counter
  (*ppCurNA)->m_pNum = (CNAPattern *) ((char *)(*ppCurNA)->m_pNum + 1) ;
}

CNAPattern *pCurNA = pNAStart ;

// again step through the tree  (and all observations)
for (i = n - 1; i >= 0; i--)
{
  pdCurVal = pdDat + i ;

  pCurNA = pNAStart ;

  // gets the according node for the current observation
  for (j = p - 1; j >= 0; j--)
  {
    if (R_IsNA (*pdCurVal))
      pCurNA = pCurNA->m_pNA ;
    else
      pCurNA = pCurNA->m_pNum ;

    pdCurVal += n ;
  }

    //  if the index array for the current NA-pattern has not yet been
    //  created
  if (!pCurNA->m_pNA)
```

```
      {  // creating index - array
        int nCount = (int) (pCurNA->m_pNum) ;
        pCurNA->m_pNA = (CNAPattern*) new int [nCount] ;
        pCurNA->m_pNum = NULL ;
      }

        // stores the index of the current observation
      ((int*) pCurNA->m_pNA) [(int) pCurNA->m_pNum] = i ;

      pCurNA->m_pNum = (CNAPattern *) ((char *)pCurNA->m_pNum + 1) ;
    }

      // recursively writes all indices from all last level-nodes to one
      //    big index array pnOrder
    WriteNAPattern (pNAStart, pnPattern, pnOrder) ;

    delete pNAStart ;
  }

  void matmult (double *pMat1, int *pnRow1, int *pnCol1, double *pMat2,
                int *pnCol2, double *pMatRet)
  {
////////////////////////////////////////////////////////////////////////
//
//  AIM:    Multiplies two matrices pMat1 and pMat2
//
//  INPUT:  pMat1   array of length *pnRow1 * *pnCol1
//          pnRow1  Number of rows of pMat1
//          pnCol1  Number of columns of pMat1
//          pMat2   array of length *pnCol1 * *pnCol2
//          pnCol2  Number of columns of pMat2
//          pMatRet double array of length *pnRow1 * *pnCol2
//                    the resulting matrix is returned by this array
//
////////////////////////////////////////////////////////////////////////

    int &nRow1 = *pnRow1 ;
    int &nCol1 = *pnCol1 ;
    int &nCol2 = *pnCol2 ;

    int i,j,h ;
    for (i = 0; i < nRow1; i++)    //  iterate through tows of pMat1
      for (j = 0; j  < nCol2; j++) //  iterate through coloumns of pMat2
      {
```

```
        double *pdCurPoint = pMatRet + i + j * nRow1 ;
        *pdCurPoint = 0 ;
                                  //  iterate through rows of pMat2
        for (h = 0; h < nCol1; h++)
          *pdCurPoint += pMat1 [i + h * nRow1] * pMat2 [h + j * nCol1] ;
      }
  }

  void pcImpDataDistance (int *pnPar, double *pdDat, double *pdImp,
                          int *pnWhich, int *pnObsOk, double *pdDistance)
  {
////////////////////////////////////////////////////////////////////////
//
//  AIM:    Calculates the distance of two matrixes (pdDat and pdImp) for
//            estimating the quality of an imputation.
//            Only elements which have been missing in the original
//            matrices are considered. Further values are ignored which are
//           "not ok", as indicated by array pnObsOk
//
//  INPUT:  pnPar      array containing parameters:
//            0:       rowNum of matrix pdDat
//            1:       colNum of matrix pdDat (not needed here)
//            2:       number of missings in original matrix
//            3:       flag indicating whether manhattan (0) or
//                     square distances (1) shall be calculated
//          pdDat      first matrix to compare
//          pdImp      second matrix to compare
//          pnWhich    indizes of elements which have been missing in the
//                       original dataset
//          pnObsOk    array specifying observations which shall be compared
//          pdDistance the resulting distance is returned via this variable
//
////////////////////////////////////////////////////////////////////////

//  the first coloumn of pnWhich must be ordered ascending in order
//    to find the right indizes for pnObsOk

    int &n = pnPar [0] ;       //  pdat and pImp are of size n x p
    int &nMiss = pnPar[2] ;    //  pnWhich is of size nMiss x 2

    int &nSqrDist = pnPar[3] ; //  Flag indicating whether manhattan (0) or
                               //    square distances (1) shall be calculated
    int i ;
```

```
int *pnWhichCol = pnWhich + nMiss ;

*pdDistance = 0 ;

   double dTempDist = 0 ;

int nLastRowIdx = *pnWhich ;
int nCurIdx ;
double dSqr ;

int nCountVals = 0 ;

for (i = 0; i < nMiss; i++)
{                          //  iterating through all missing values
  if (nLastRowIdx != pnWhich[i])
  {  //  all distances of the current observation have been added
    if (nSqrDist)
    {
      *pdDistance += sqrt (dTempDist) ;
      dTempDist = 0 ;
    }
    pnObsOk ++ ;
  }
  nLastRowIdx = pnWhich[i] ;
  if (!*pnObsOk )  //  corresponding observation is outlier - do not
                   //    include in distance calculation
    continue ;
                   // calculating index for current observation
  nCurIdx = pnWhich[i] - 1 + (pnWhichCol[i] - 1) * n ;
  if (nSqrDist)    // when euclidean distance should be calculated:
  {                // square distance & add to dTempDist
    dSqr = pdDat[nCurIdx] - pdImp[nCurIdx] ;
    dTempDist += dSqr * dSqr ;
  }
  else            // otherwise: add distance to dTempDist
    dTempDist += fabs (pdDat[nCurIdx] - pdImp[nCurIdx]) ;

  nCountVals ++ ;
}

if (nSqrDist)
  dTempDist = sqrt (dTempDist) ;
*pdDistance += dTempDist ;
*pdDistance /= nCountVals ;
```

```
  }

  void iterproj_ll_one (int p, int naccess, int nM_, int *pnNAIdx,
                        double *pdData, double *pdProjMat, double *pdTemp_p,
                        int nIterMax)
  {
////////////////////////////////////////////////////////////////////////
//
//  AIM:    projects one observation iterativel onto a set of princiapl
//          components, given by pdProjMat = gamma[,1:k] %*% t(gamma[,1:k])
//
//  INPUT:  p         dimensionality of data space
//          naccess   rowNum of the underlying matrix containing the
//                       considered observation (for addressing only)
//          nM_       number of missing values
//          pnNAIdx   array containing indices of missing values
//          pdData    Data matrix, starting with the considered observation
//                      the imputed values are directly written into this
//                      matrix
//          pdProjMat projection matrix - basically consisting of loadings
//                      = gamma[,1:k] %*% t(gamma[,1:k])
//          pdTemp_p  array of length p - used for buffering results
//          nIterMax  maximum number of iterations
//
////////////////////////////////////////////////////////////////////////

    int j, h ;
    double dDist ;

    double *pdCurData ;
    double dConvergenceThreshold = 0.001 * nM_ ;

    double *pdCurProjMat ;

    int nCurIter ;

          // checking all variables for NAs and replacing them with 0
    for (j = nM_ - 1; j >= 0; j--)
      if (R_IsNA (pdData[pnNAIdx[j] * naccess]))
        pdData[pnNAIdx[j] * naccess] = 0 ;

    dDist = 1 ;

    nCurIter = 0 ;
```

```
    while (dDist > dConvergenceThreshold && nCurIter++ < nIterMax)
    {
      pdCurProjMat = pdProjMat ;
          //  performing Matrix multiplication
          //   pdTemp_p = curobs %*% pdProjMat
      for (j = 0; j < nM_; j++)
      {
        pdCurData = pdData ;
        double &dCurEst = pdTemp_p[j] = 0 ;

        pdCurProjMat = pdProjMat + pnNAIdx[j] * p ;

        for (h = 0; h < p; h++)    //  Matmult - jeweils eine spalte
        {
          dCurEst += *pdCurData * *pdCurProjMat ++ ;
          pdCurData += naccess ;
        }
      }

      //  calculates distance & writes estimations vor missing values
      //    back to data matrix
      dDist = 0 ;

      for (j = 0; j < nM_; j++)
      {
        double &dCurPos = pdData[pnNAIdx [j] * naccess] ;
        dDist += fabs (dCurPos - pdTemp_p[j]) ;
        dCurPos = pdTemp_p[j] ;
      }
    }
  }

  void iterproj (int *pnPar, double *pdDat, double *pdPC, int *pnNAPattern,
                 int *pnNAOrder, int *pnI)
  {
////////////////////////////////////////////////////////////////////////////
//
//  AIM:    projects n observations iteratively onto k principal components
//          given by pdPC, by calling iterproj_ll_one
//
//  INPUT:  pnPar       array containing parameters:
//            0:        n := rowNum of matrix pdDat
//            1:        p := colNum of matrix pdDat (not needed here)
```

```
//            2:        k := number of relevant PCs
//          pdDat       Data Matrix containing observations to project (nxp)
//          pdPC        Matrix containing loadings  (Gamma) (pxp)
//          pnNAPattern array of length n, containing an ID of the
//                        NA-pattern of each observation. Equal ID means
//                        NA-patterns are ident, different IDs: NA-patterns
//                        differ.
//          pnNAOrder   Index - array which sorts observations in a way,
//                      that observations with equal NA-pattern are blocked
//          pnI         array of size n*p indicating whether a values has
//                        been observed (1) or was missing (0)
//
////////////////////////////////////////////////////////////////////////////

    int &n = pnPar [0] ;
    int &p = pnPar [1] ;
    int &k = pnPar [2] ;

    int i, j, h;

    double *pdCurDat ;

    int *pnNAIdx = new int [p], *pnNotNAIdx = new int [p] ;
    double *pdTemp = new double [p] ;

    int nLastNaPattern = pnNAPattern[0] - 1 ;

    double *pdProj = new double [p * p];

    int nCurIdx, nM_ = 0 , nM = 0 ;

    int *pnCurI ;

    double *pdGTG = new double [p * p];

    {  //  Calculate gamma [,1:k] %*% t(gamma[,1:k]) for iterproj_ll_one
      for (i = 0; i < p; i++)
        for (j = 0; j < p; j++)
        {
          double &dCurGTG = pdGTG[j + i * p] = 0 ;
          for (h = 0; h < k; h++)
            dCurGTG += pdPC[j + h * p] * pdPC[i + h * p] ;
        }
    }
```

```
    for (i = 0; i < n; i++)
    {  // for each row of pdDat
      nCurIdx = pnNAOrder [i] ;  // position of current observation

      if (nLastNaPattern != pnNAPattern[i])
      {  // if the NA-pattern has changed..
        nLastNaPattern = pnNAPattern[i] ;
        nM_ = nM = 0 ; //  nM ... number of not missings in current pattern

        pdCurDat = pdDat + nCurIdx ;
        pnCurI = pnI + nCurIdx ;
         //  get NA-pattern (which observations were missing)
         //    out of I
        for (j = 0; j < p; j++)
        {
          if (!*pnCurI)//R_IsNA (*pdCurDat))
            pnNAIdx [nM_++] = j ;
          else
            pnNotNAIdx[nM ++] = j ;

          pdCurDat += n ;
          pnCurI += n ;
        }
      }
        // call iterproj_ll_one (for each observations)
      iterproj_ll_one (p, n, nM_, pnNAIdx, pdDat + nCurIdx, pdGTG, pdTemp,
        pnPar[3]) ;
    }

    delete [] pdGTG ;
    delete [] pnNAIdx ;
    delete [] pnNotNAIdx ;
    delete [] pdTemp ;
    delete [] pdProj ;
  }

  void mahaproj (int *pnPar, double *pdDat, double *pdPCs, double *pdSdev, int *pnNAP
  {
////////////////////////////////////////////////////////////////////////
//
//  AIM:    implements the Mahalanobis distance based projection method
//
//  INPUT:  pnPar       array containing parameters:
```

```
//            0:        n := rowNum of matrix pdDat
//            1:        p := colNum of matrix pdDat (not needed here)
//            2:        k := number of relevant PCs
//          pdDat       Data Matrix containing observations to project (nxp)
//          pdPCs        Matrix containing loadings  (Gamma) (pxp)
//          pdSdev      array of length p convaining the eigenvalues ^0.5
//          pnNAPattern array of length n, containing an ID of the
//                        NA-pattern of each observation. Equal ID means
//                        NA-patterns are ident, different IDs: NA-patterns
//                        differ.
//          pnNAOrder   Index - array which sorts observations in a way,
//                      that observations with equal NA-pattern are blocked
//          pnI         array of size n*p indicating whether a values has
//                        been observed (1) or was missing (0)
//
////////////////////////////////////////////////////////////////////////

    int &n = pnPar[0] ;
    int &p = pnPar[1] ;
    int &k = pnPar[2] ;

    int *pnipiv = new int [p] ;
    int nFoo ;

    int nCurIdx, nLastNaPattern = *pnNAPattern -1;

    int *pnNAIdx = new int [p], *pnNotNAIdx = new int [p] ;

    double *pdGS = new double [p * p] ;  //  G %*% Sigma^-1
    double *pdProj = new double [p * p] ;
    double *pdInv = new double [p * p] ;
    double *pdGSG = new double [p * p] ;

    int *pnCurNAPattern = new int [p] ;

    double *pdCurDat ;
    int nM ;   //  # of not missing in the current observation
    int nM_ ;  //  # of missing in the current observation

    double *pdSigmaInv = new double [p] ;
    double *pdSigmaOne = new double [p] ;

    double *pdSigmaUsed ;
```

```
int i, j, h, l ;

int *pnPtP = new int [p] ;    //  p times p

for (i = 0; i < p; i++)
{
  pnPtP[i] = p * i ;
  pdSigmaOne[i] = i >= k ;
  if (pdSdev[i] < 10e-12)
    pdSigmaInv[i] = 10e12 ;
  else
    pdSigmaInv[i] = 1/(pdSdev[i] * pdSdev[i]) ;
}

double *pdNewObs = new double [p] ;

double *pdCurGS ;
double *pdCurPC ;
double *pdCurProj ;
double *pdCurInv ;
double *pdCurGSG ;

int nMode = 0 ;
int nkUsed ;

int *pnCurI ;


for (i = 0; i < n; i++)
{
  nCurIdx = pnNAOrder [i] ;

  if (nLastNaPattern != pnNAPattern[i])
  {  //  NA-pattern has changed -> calculate new projection matrizes
    pdCurDat = pdDat + nCurIdx ;
    pnCurI = pnI + nCurIdx ;
    nLastNaPattern = pnNAPattern [i] ;

    nM_ = nM = 0 ;

    //  get an index array for missing and the not missing values
    for (j = 0; j < p; j++)
    {
      if (pnCurNAPattern[j] = !(*pnCurI))
```

```
          pnNAIdx [nM_++] = j ;
        else
          pnNotNAIdx[nM ++] = j ;

        pdCurDat += n ;
        pnCurI += n ;
      }

      if (nM_ == 0)
      {  // no missing values
        nMode = 0 ;
        continue ;
      }
      else if (nM >= k)
//  over-determined - 2 hyperplanes which do not cross - calculate
//    minimum distance between hyperplanes
      {   //  Mahalanobis algorithm may be applied, but may be unstable
          //  instead of using diag (Sigma), use 0,0,0..,0,1,...,1
          //  which results in the original minimum distance projection
        nMode = 1 ;
        pdSigmaUsed = pdSigmaOne ;

        pdCurGS = pdGS ;
        pdCurInv = pdInv ;

            //  GS = G[!m,!k] %*% t(G[!m,!k])
        for (j = 0; j < nM_; j++)
          for (h = 0; h < nM_; h++)
          {
        //  filling diagonal matrix  into pdInv - mat (for mat inversion)
            *pdCurInv++ = j == h ;
            double &dCurGS = *pdCurGS++ = 0 ;
            for (l = k; l < p; l++)
              dCurGS += pdPCs [pnNAIdx [j] + pnPtP[l]] *
                        pdPCs [pnNAIdx [h] + pnPtP[l]] ;
          }

            //  pdInv = GS^-1
        F77_CALL(dgesv)(&nM_, &nM_, pdGS, &nM_, pnipiv, pdInv, &nM_, &nFoo) ;

        pdCurGS = pdGS ;

            //  GS = G[m,!k] %*% t(G[!m,!k])
        for (h = 0; h < nM_; h++)
```

```
    for (j = 0; j < nM; j++)
    {
      double &dCurGS = *pdCurGS++ = 0 ;
      for (l = k; l < p; l++)
        dCurGS += pdPCs [pnNotNAIdx [j] + pnPtP[l]] *
                  pdPCs [pnNAIdx [h] + pnPtP[l]] ;
    }

  matmult(pdGS, &nM, &nM_, pdInv, &nM_, pdProj) ;

  nFoo = nM_ * nM ;
}
else
{      // exactly defined (nM_ == k) or under-determined
       //  - find point with minimum Mahalanobis distance.
       //  Apply Mahalanobis algorithm (minimize mahalanobis distance)

  nMode = 2 ;
  nkUsed = p ;  //   this projection only works with all dimension

 pdSigmaUsed = pdSigmaInv ;

    //  pdGS = G %*% Sigma^-1
  pdCurGS = pdGS ;
  for (j = 0; j < nkUsed; j++)
  {
    pdCurPC = pdPCs + pnPtP[j] ;
    double &dCurSigmaInv = pdSigmaUsed [j] ;
    for (h = 0; h < nM_; h++)
      *pdCurGS++ = pdCurPC[pnNAIdx [h]] * dCurSigmaInv ;
  }

   //  pdProj = pdGS %*% t(G)   (nM_ x nkUsed) x  (nkUsed x nM_)
  pdCurProj = pdProj ;
  pdCurInv = pdInv ;


  for (j = 0; j < nM_; j++)
  {
    for (h = 0; h < nM_; h++)
    {
      *pdCurInv++ = h == j ;
  //  filling diagonal matrix  into pdInv - mat (for mat inversion)
```

```
        double &dCurProj = *pdCurProj ++ = 0 ;

        for (l = 0; l < nkUsed; l++)
          dCurProj += pdGS [nM_ * l + h] * pdPCs[pnNAIdx [j] + pnPtP[l]] ;
      }
    }

    F77_CALL(dgesv)(&nM_, &nM_, pdProj, &nM_, pnipiv, pdInv, &nM_, &nFoo) ;

    {    // GSG = GS %*% Gamma [,1:k]
      pdCurGSG = pdGSG ;

      for (j = 0; j < nM_; j++)
        for (h = 0; h < p; h++)
        {
          double &dCurGSG = pdGSG[j+h * nM_] = 0 ;//*pdCurGSG++ = 0 ;
          for (l = 0; l < nkUsed; l++)
            dCurGSG += pdGS[j + l * nM_] * pdPCs[h + pnPtP[l]] ;
        }
    }

    //  pdProj = pdInv %*% pdGSG
    matmult (pdInv, &nM_, &nM_, pdGSG, &p, pdProj) ;
    nFoo = nM_ * p  ;
  }

  InvertSgn (pdProj, &nFoo) ;

}

if (nMode == 1)
{  // over determined

        // multiplies current observation with pdProj[M,]
  pdCurDat = pdDat + nCurIdx ;
  for (j = 0; j < nM_; j++)
  {
    double &dCurNewObs = pdNewObs[j] = 0 ;

    for (h = 0; h < nM; h++)
      dCurNewObs += pdCurDat [pnNotNAIdx [h] * n] * pdProj [j * nM + h] ;
  }
  //  write imputed values back to data matrix
  for (j = 0; j < nM_; j++)
```

```
        pdCurDat [pnNAIdx [j] * n] = pdNewObs [j] ;
    }
    else if (nMode == 2)
    {  // under determined (or exactly)

          // multiplies current observation with pdProj[M,]
      for (h = 0; h < nM_; h++)
      {
        pdCurDat = pdDat + nCurIdx;
        pdCurProj = pdProj + h ;
        double &dCurNewObs = pdNewObs[h] = 0 ;
        for (j = 0; j < p; j++)
        {
          if (!pnCurNAPattern [j])
            dCurNewObs += *pdCurDat * *pdCurProj ;
          pdCurProj += nM_ ;
          pdCurDat += n ;

        }
      }
            //  write imputed values back to data matrix
      pdCurDat = pdDat + nCurIdx ;
      for (h = 0; h < nM_; h++)
        pdCurDat [pnNAIdx [h] * n] = pdNewObs [h] ;
    }
  }

  delete [] pnNAIdx ;
  delete [] pnNotNAIdx ;

  delete [] pdGS ;
  delete [] pdProj ;
  delete [] pdInv ;
  delete [] pnipiv ;
  delete [] pdSigmaOne ;
  delete [] pdSigmaInv ;
  delete [] pdNewObs ;
  delete [] pnCurNAPattern ;
  delete [] pdGSG ;
  delete [] pnPtP ;

}

void scale (int *pnPar, double *pdDat, double *pdMean, double *pdScale)
```

```
  {
////////////////////////////////////////////////////////////////////////
//
//  AIM:    Scales a data-matrix pdDat according to center Vector pdMean
//          and scale vector pdScale
//
//  INPUT:  pnPar   array containing parameters:
//            0:    n := rowNum of matrix pdDat
//            1:    p := colNum of matrix pdDat (not needed here)
//            2:    bAdd := flag whether z-Transformation (1) or
//                  inverse z-Transformation should be done
//          pdDat   array of length (n*p) representing the data matrix
//          pdMean  mean vector (array of size p)
//          pdScale scale vector (array of size p)
//
////////////////////////////////////////////////////////////////////////

    int &n = pnPar[0] ;
    int &p = pnPar[1] ;
    int &bAdd = pnPar[2] ;
    int i, j ;

    if (bAdd)        //  if the data should be centered
      for (j = 0; j < p; j++)
      {   // for each variable
        double &dMean = pdMean[j] ;
        double &dScale = pdScale[j] ;
          // for each observation
        for (i = n - 1; i >= 0; i--)
          *pdDat++ = (*pdDat - dMean) / dScale ;
      }
    else              //  if the data is already centered and the resulting
                // dataset should have center pdCenter and sd = pcScale
      for (j = 0; j < p; j++)
      {
        double &dMean = pdMean[j] ;
        double &dScale = pdScale[j] ;
          // for each observation
        for (i = n - 1; i >= 0; i--)
          *pdDat++ = (*pdDat * dScale) + dMean ;
      }
  }

  void InvertSgn (double *pd, int *pn)
```

```
  {
////////////////////////////////////////////////////////////////////////
//
//  AIM:  Inverts the signs of array pd of length pn
//
//  INPUT:  pd  array containing values which signs shall be changed
//          pn  points to an int containing the number of elements of pd
//
////////////////////////////////////////////////////////////////////////

    int &n = *pn, i ;
    for (i = n; i > 0; i--)
      *pd++ = _chgsign (*pd) ;
  }
```

Bibliography

[1] Rousseeuw, P.J., van Driessen, K., (1999) A fast algorithm for the minimum covariance determinant estimator. Technometrics, 41, 212–223.

[2] Croux, C., Filzmoser, P., Oliveira, M.R., (2007) Algorithms for projection-pursuit robust principal component analysis. Chemometrics and Intelligent Laboratory Systems, To appear.

[3] Croux, C., Ruiz-Gazen, A., (2005) High Breakdown Estimators for Principal Components: The Projection-pursuit Approach Revisited. Journal of Multivariate Analysis, 95, 206-226.

[4] Rousseeuw, P.J., Croux, C., (1993) Alternatives to the Median Absolute Deviation. Journal of The American Statistical Association, 88, 1273-1283.

[5] Rousseeuw, P.J., Leroy, A.M., (1987) Robust Regression and Outlier Detection. Wiley, New York.

[6] Walczak, B., Massart, D.L., (2001) Dealing with missing data. Part I. Chemometr. Intell. Lab. Syst., 58, 15-27.

[7] Serneels, S., Verdonck, T., (2007) Principal component analysis for data containing outliers and missing elements. Unpublished manuscript.

[8] Krzanowski, W.J., (1979) Between-Groups Comparison of Principal Components. Journal of the American Statistical Association, 74, 703–707.

[9] Hubert, M., Rousseeuw, P.J., Vanden Branden, K., (2005) ROBPCA: a New Approach to Robust Principal Component Analysis. Technometrics, 47, 64-79.

[10] Little, R.J.A, Rubin, D.B., (2002) Statistical Analysis with Missing Data. Wiley, New York.

[11] Hastie, T., Tibshirani, R., Sherlock, G., Eisen, M., Brown, P., Botstein, D., (1999) Imputing Missing Data for Gene Expression Arrays. Stanford University Statistics Department, Technical report.

[12] Gabriel, K.R., (1971) The biplot graphical display of matrices with applications to principal component analysis. Biometrika, 58, 453–467.

[13] Hotelling, H., (1933) Analysis of a complex of statistical variables into principal components. Journal of Educational Psychology, 24, 417-441.