Multiple Outlier Detection and Cluster Analysis of Multivariate Normal Data

Geoffrey Robson*

Thesis presented in partial fulfilment of the requirements for the degree of
Master of Engineering Science at the University of Stellenbosch

Supervisors: Prof. B.M. Herbst and Dr. N.L. Muller
Department of Applied Mathematics
University of Stellenbosch
December 2003

*The financial assistance of the National Research Foundation (NRF) towards this research is hereby acknowledged. Opinions expressed and conclusions arrived at, are those of the author and are not necessarily to be attributed to the NRF.

Declaration

I, the undersigned, hereby declare that the work contained in this thesis is my own original work and that I have not previously in its entirety or in part submitted it at any university for a degree.

Signature _____ Date _____

Abstract

Outliers may be defined as observations that are sufficiently aberrant to arouse the suspicion of the analyst as to their origin. They could be the result of human error, in which case they should be corrected, but they may also be an interesting exception, and this would deserve further investigation.

Identification of outliers typically consists of an informal inspection of a plot of the data, but this is unreliable for dimensions greater than two. A formal procedure for detecting outliers allows for consistency when classifying observations. It also enables one to automate the detection of outliers by using computers.

The special case of univariate data is treated separately to introduce essential concepts, and also because it may well be of interest in its own right. We then consider techniques used for detecting multiple outliers in a multivariate normal sample, and go on to explain how these may be generalized to include cluster analysis.

Multivariate outlier detection is based on the Minimum Covariance Determinant (MCD) subset, and is therefore treated in detail. Exact bivariate algorithms were refined and implemented, and the solutions were used to establish the performance of the commonly used heuristic, Fast–MCD.

Opsomming

Uitskieters word gedefinieer as waarnemings wat tot só 'n mate afwyk van die verwagte gedrag dat die analis wantrouig is oor die oorsprong daarvan. Hierdie waarnemings mag die resultaat wees van menslike foute, in welke geval dit reggestel moet word. Dit mag egter ook 'n interressante verskynsel wees wat verdere ondersoek benodig.

Die identifikasie van uitskieters word tipies informeel deur inspeksie vanaf 'n grafiese voorstelling van die data uitgevoer, maar hierdie benadering is onbetroubaar vir dimensies groter as twee. 'n Formele prosedure vir die bepaling van uitskieters sal meer konsekwente klassifisering van steekproefdata tot gevolg hê. Dit gee ook geleentheid vir effektiewe rekenaar implementering van die tegnieke.

Aanvanklik word die spesiale geval van eenveranderlike data behandel om noodsaaklike begrippe bekend te stel, maar ook aangesien dit in eie reg 'n area van groot belang is. Verder word tegnieke vir die identifikasie van verskeie uitskieters in meerveranderlike, normaal verspreide data beskou. Daar word ook ondersoek hoe hierdie idees veralgemeen kan word om tros analise in te sluit.

Die sogenaamde *Minimum Covariance Determinant* (MCD) subversameling is fundamenteel vir die identifikasie van meerveranderlike uitskieters, en word daarom in detail ondersoek. Deterministiese tweeveranderlike algoritmes is verfyn en geïmplementeer, en gebruik om die effektiwiteit van die algemeen gebruikte heuristiese algoritme, *Fast–MCD*, te ondersoek.

Contents

List of Figures

List of Tables

List of Symbols and Acronyms

α	significance probability (p 3)
g	number of groups (clusters) in a sample
h	initial inlier subset size (p 4)
I_n	inlier subset of size n (p 4)
κ	maximum proportion of contamination (p 4)
k	maximum number of outliers (p 4)
l	level in the subset tree (p 37)
m	number of restarts used for Fast-MCD (p 63)
n	size of inlier subset under consideration
N	size of complete sample
Ω	lower bound on the order of complexity of an algorithm
p	multivariate dimension
ρ	global convergence probability (p 64)
s^2	unbiased estimate of variance of I_n
$\hat{s}^2$	MLE of variance of I_n
$\mathbf{S}$	MLE of covariance matrix of I_n
x_i	ith observation of sample
$x_{(i)}$	ith smallest observation
$\bar{x}$	univariate mean of I_n
$\mathbf{x}_i$	$p \times 1$ vector of ith observation of a multivariate sample
$\bar{\mathbf{x}}$	$p \times 1$ mean vector of I_n
z_i	Mahalanobis distance of $\mathbf{x}_i$ (p 73)
$z_{(n)}$	maximum z_i used as test statistic

EDR	Extreme Deviate Removal
ESD	Extreme Studentized Deviate
LTS	Least Trimmed Squares
MCD	Minimum Covariance Determinant
-BB	Branch and Bound algorithm
-BF	Bernholt and Fischer's algorithm
-EXT	Extreme algorithm

-SW	Sweepline algorithm
MLE	Maximum Likelihood Estimate
MVAR	Minimum Variance
pdf	probability density function

Chapter 1

Introduction

1.1 Definitions and motivation

The concept of an outlier is as old as data analysis itself. One dictionary* uses the phrase "detached from the main mass" to describe the term, but in statistical parlance, a refinement of this definition recognizes two types of outlier. One is an observation that, though extreme, originates from the same distribution as that which produced the bulk of the data. This *extreme observation* does not harm the integrity of the data. In fact, its inclusion is important so as to ensure that useful information is not discarded. The more insidious outlier is one that does not belong to the same distribution as the bulk of the data, but is the result of human error or the presence of a separate generating mechanism and a different distribution. This outlier is termed a *contaminant* [2]. In this thesis the focus is solely on outliers of this type, so the two terms are used interchangeably. It is usually only in the case of heavy tailed distributions, such as Student's t, that outlying observations are not contaminants. This thesis deals only with contamination of samples derived from a normal distribution, which is not *outlier-prone* [13] (p 1). It is also assumed that nothing is known *a priori* about the parameters of the distribution, which is commonly the case.

Rousseeuw and Leroy [25] (p vii) point out that "Outliers occur very frequently in real data, and they often go unnoticed because nowdays (*sic*) much data is processed by computers, without careful inspection or screening." Fukunaga [9] (p 235) says that "It is widely believed in the pattern recognition field that classifier performance can be improved by removing outliers..." This field (see Bishop [4] and Fukunaga

*The online Oxford English Dictionary `http://dictionary.oed.com/`

[9]) relies on topics like density estimation, discriminants, and the reduction of dimensionality that all depend on the multivariate normal distribution to some extent. The many strengths of the multivariate normal distribution are summarized in [4] (p 37). One of these is that using it is often a matter of applying standard techniques from linear algebra, yet as with the univariate distribution, a sound theoretical basis is provided by the central limit theorem. As a result, many multivariate data sets (particularly biometric ones) are modelled accurately as multivariate normal.

If the outliers in a sample are themselves well grouped, and perhaps even normally distributed, then the problem becomes one of cluster analysis rather than outlier detection. Cluster analysis is broadly defined as the partitioning of N observations into g groups, such that similar observations fall into the same group. It is thus a large and varied field, so we defer a more detailed consideration of it until section 1.6. The emphasis is typically confined to *categorizing* rather than *characterizing* the data, so the distribution of the data is secondary in importance to the groups it consists of. It is not traditionally thought of as a type of outlier problem, but we believe that generalizing the problem into one which treats the data as being composed of an unknown number of clusters, with those observations not assigned to a cluster regarded as outliers, is potentially very useful. Such a unified approach combines the well understood statistical tests from outlier detection with the emphasis on efficient algorithms from cluster analysis, to produce a flexible tool for the analysis of multivariate data.

The use of computers for the automatic acquisition and storage of data easily results in very large data sets. It becomes easier to record not only more observations, but more variables too, so the complexity of the data also increases. This leads us to the field of *data mining*, which is defined by Rocke [21] as the exploration of massive data sets. It is typically associated with the search for correlations or relationships between variables, and also for groups of related data. Our suggested approach therefore fits neatly within the framework of this field.

It is important that a distinction be made between data that are supposed to have arisen from a probability model, and those which are assumed to have a definite underlying structure that stems from a functional relationship amongst their variables. If the stochastic nature of these two models was removed, then in the first case one would simply obtain a set of identical observations whereas in the second, the data would be a perfect reflection of the function of the variables, for instance a straight line. Regression and time-series analysis fall into this structured data analysis category, for which a multivariate normal model is not suitable.

1.2 A single outlier in univariate data

A standard application of inferential statistics will typically involve a relatively small number of observations. The analyst has a good idea of what distribution the data will follow, and may go about confirming this before estimating one or more of its parameters. Outlier detection (if it takes place at all), is usually in the form of visual inspection of a suitable plot. If an observation is suspiciously outlying, then it is probably discarded without much thought. Despite this being rather informal, it is simple and effective.

A sample contaminated with outliers may be viewed as a departure from the expected model, so a more formal approach to outliers could involve a parametric goodness-of-fit test (which might already have been carried out above when confirming the suspected distribution) on the sample. Specifically, the sample coefficient of kurtosis* is "very effective for detecting the presence of outliers in a normal sample." [13] (p 6). While such a test "may tell one that the sample contains one or more outliers,... it does not identify any specific observation as an outlier." [13] (p 14).

To achieve this, a specific observation may be tested for discordancy. For univariate data it is natural to test the observation that is furthest from the bulk of the data, i.e., the one with greatest distance from the mean. Normalizing this distance by dividing by the sample standard deviation gives one a statistic that can be tested under the null hypothesis that no outliers are present. Only in special cases is the null distribution of such a statistic tractable, and even then, it can often only be expressed in the form of recurrence relations or infinite series. Accurate approximations of the *exceedence* or *significance probability* of the statistic is therefore an important part of hypothesis testing. This is the probability that the test statistic takes on a value that is "more indicative of discordancy" [2] (p 115) than a certain *critical value*, and is denoted by α. This is the cut-off value for classifying the observation as either an inlier or outlier. An acceptable Type I error (incorrectly classifying an inlier as an outlier) is decided upon to calculate the critical value. The reader may be concerned about the inclusion of the potential outlier in the calculation of the sample mean and standard deviation used in the test statistic. In turns out however, that in this and many other cases, the *inclusive* and *exclusive* statistics are equivalent. See [2] (pp 108-109) and [27] (chap 2, pp 6-7).

*The reader is reminded that this is the ratio of the fourth and second squared sample central moments. Its null distribution is intractable, but tables of critical values have been compiled [2] (p 231).

1.3 Multiple outliers in univariate data

If only one outlier is present, then a suitable discordancy test should easily identify it. The disadvantage of these tests lies in their sensitivity to what is known as *masking* if more than one outlier is present. "If the most extreme observation is compared with the rest of the sample and more than one outlier is present, then some of these outliers may well [cause it to be] compared against too widely dispersed a subset so that its extremeness is not apparent." [27] (chap 3, p 1).

The discordancy tests for a single observation can be extended to test a group of observations in an attempt to avoid the masking effect. A simple example of such a *block* test statistic is the distance from the minimum to the maximum observations, again normalized by the sample standard deviation.* This tests for an upper and lower outlier pair, and if the statistic exceeds the chosen critical value, then both observations are classified as outliers. The block procedures are limiting in that they assume that not only the number of outliers, but also their configuration, is known. The test statistic given above, for instance, would be useless if both contaminants were upper outliers. Likewise, if only one outlier was present, then the test could not detect it alone.

A situation where a sample contains either a specified number of outliers or none at all is artificial. The best we can expect is an upper bound k on the number of outliers present. This may be calculated as $k = \lceil \kappa N \rceil$, where κ is the maximum proportion of contamination that the data are presumed to contain, and N the complete sample size. We may then perform k consecutive tests, with each testing for one to k outliers.

Because the configuration of the outliers is unknown, one must begin each test by deciding which observations are most suspicious, and hence to be treated as the potential outliers. The remaining observations make up the *inlier subset* and we denote one of size n by I_n. Because one outlier is tolerable, the inlier subsets range in size from $h = N - k + 1$ (when testing for k outliers) to the complete sample size N (when testing for a single outlier).

One way of finding inlier subsets is to remove the most deviant observation from the sample $k - 1$ times, so that $I_N \supset I_{N-1} \supset \ldots \supset I_{h+1} \supset I_h$. As an alternative to this sequential removal approach, we could construct each of the k inlier subsets independently of each other by considering all $\binom{N}{n}$ subsets, and setting I_n equal to whichever one is optimal according to some test statistic. This might appear to

*This is test **N6** in [2] (p 226).

be computationally impractical for samples of even moderate size, but it will be seen (for example in section 2.1) that it need not entail an exhaustive search of all combinations.

Once inlier subsets have been chosen, the hypothesis testing can begin. This was originally done in an *inward* fashion by first testing the complete sample I_N for a single outlier. If it failed the test, then the sample was presumed to contain at least one outlier, and the same test was applied to I_{N-1}. This continued until the subset was judged free of outliers. Implicit in this method is the assumption that if a hypothesis test concludes that no single outlier is present, then one can infer that it is entirely free of outliers. The effect of masking invalidates this assumption, so the conclusion may well occur prematurely, resulting in a sample that still contains outliers [13] (p 51).

An alternative is to start with the smallest subset I_h that contains at most one outlier, and move *outward*, successively testing subsets of increasing size for a single outlier, and stopping when one fails the test. The justification for this form of hypothesis testing is described in chapter 3, but suffice it to say that it is derived from an appropriate definition of a Type I error, and that if the initial inlier subset does indeed have no more than one outlier, then the problem of masking is eliminated. Outward testing is thus the only technique considered in this thesis. We have chosen the terminology of Barnett and Lewis [2]. Viljoen [27] calls the inward and outward techniques *step-down* and *step-up*, whereas Hawkins [13] refers to them as *backward elimination* and *forward selection*, respectively. Also, this form of successive testing is sometimes called *sequential* testing.

1.4 A single outlier in multivariate data

The detection of multivariate outliers is substantially more problematic than univariate ones. An obvious difficulty is the graphical representation of high dimensional data. For dimensions greater than two the analyst may examine scatter plots of every possible pair of variables [15] (p 205), but this does not guarantee that outliers will be exposed. For example, an outlier containing mild but systematic errors in all of its components will remain hidden unless a suitable linear transformation of the data is first performed. The number of scatter plots is $\binom{p}{2} = \frac{1}{2}p(p-1)$, so the work required in examining them grows quadratically with the dimension of the data. Having to examine an arbitrary number of these sets of plots for various transformations of the data is clearly impractical for all but low dimensional data.

Almost any investigation of a multivariate data set will include a plot of the ordered Mahalanobis distances*. This might be used in conjunction with scatter plots to verify the normality of the data and to reveal the presence of outliers. A hypothesis test on the maximum Mahalanobis distance using the critical values tabulated by Wilks [29], can reliably detect a solitary outlier [2] (p 288). As in the univariate case, complications arise when more than one outlier is present.

1.5 Multiple outliers in multivariate data

Unlike the univariate case, where ordering the data was independent of the measure of its dispersion, the reliability of the ordering based on Mahalanobis distances depends on a measure of location and dispersion that is representative of the good data. The masking effect can very easily conceal the outliers, and if they form a cluster they can distort these estimates to such an extent that inliers appear to be outlying. This is the opposite of masking, and is known as *swamping*. The vulnerability to these negative effects is due to the fact that an "outlier can distort not only measures of location and scale but also those of orientation (i.e. correlation)." [2] (p 298).

This is illustrated in Figure 1.1, which shows a scatter plot of two bivariate normal clusters. The large ellipse marks the 95% confidence region using the sample mean and covariance matrix derived from the complete sample of 130 observations. The resulting Mahalanobis distances are shown in Figure 1.2, where swamping has produced an ostensible outlier.

If we now discard the outliers, then the smaller ellipse shown in the scatter plot is produced, which fits the distribution of the remaining inliers very well. Their Mahalanobis distances are shown in Figure 1.3, where it is apparent that the "outlier" seen in Figure 1.2 was spurious. Apart from this observation, notice how innocuous the plot in Figure 1.2 plot is, and how well the real outliers are masked.

A way of measuring the location and dispersion that is resistant to the presence of outliers is needed. The study of such *robust estimates* has tended to be treated separately from outlier *identification* techniques. Indeed, Barnett and Lewis also call them *accommodation* procedures, which suggests that one has no intention of even trying to remove them! Viljoen [27] (chap 1, p 2) draws attention to research that suggests that robust procedures are "not superior to the approach of outlier

*The definition of this distance varies somewhat, and we use that given by (5.1). It is sometimes called the Mahalanobis *squared* distance.

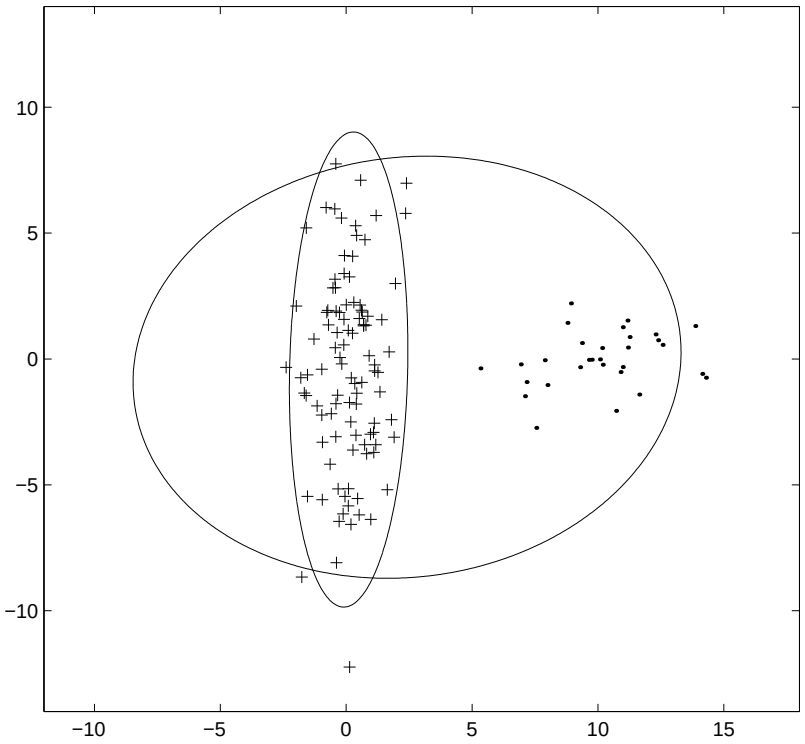


Figure 1.1: Scatter plot of 100 inliers (+) and 30 clustered outliers (·)

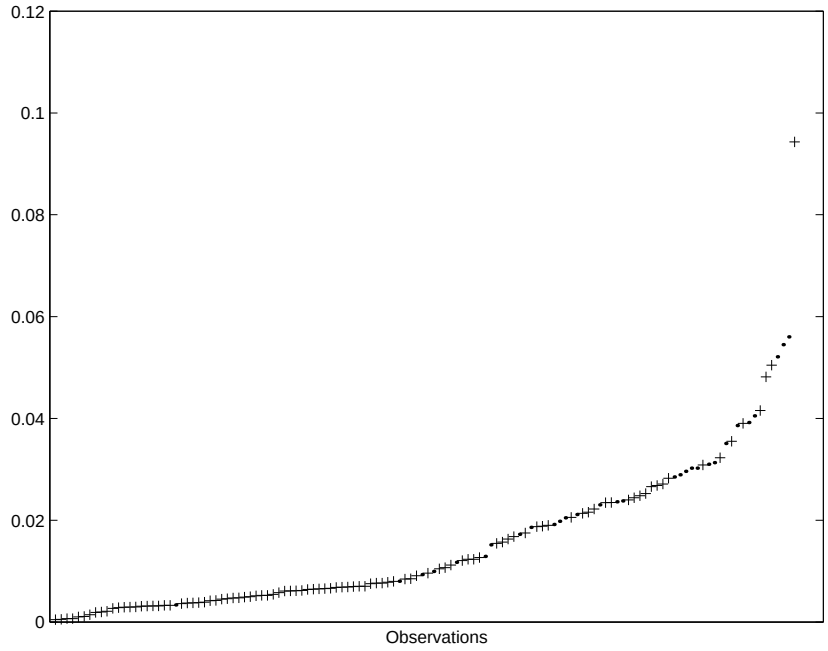


Figure 1.2: Ordered Mahalanobis distances of the entire sample of Figure 1.1

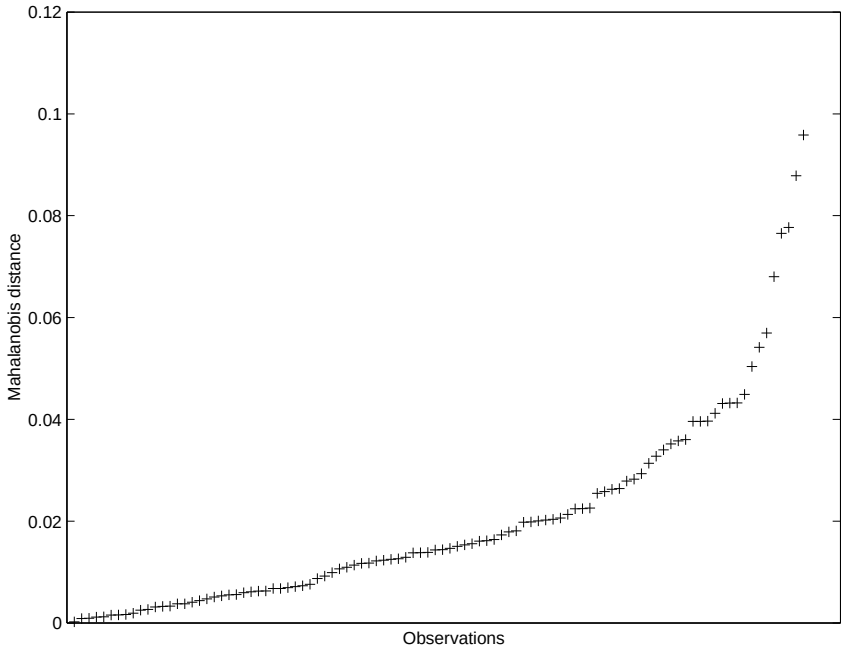


Figure 1.3: Ordered Mahalanobis distances of the inliers

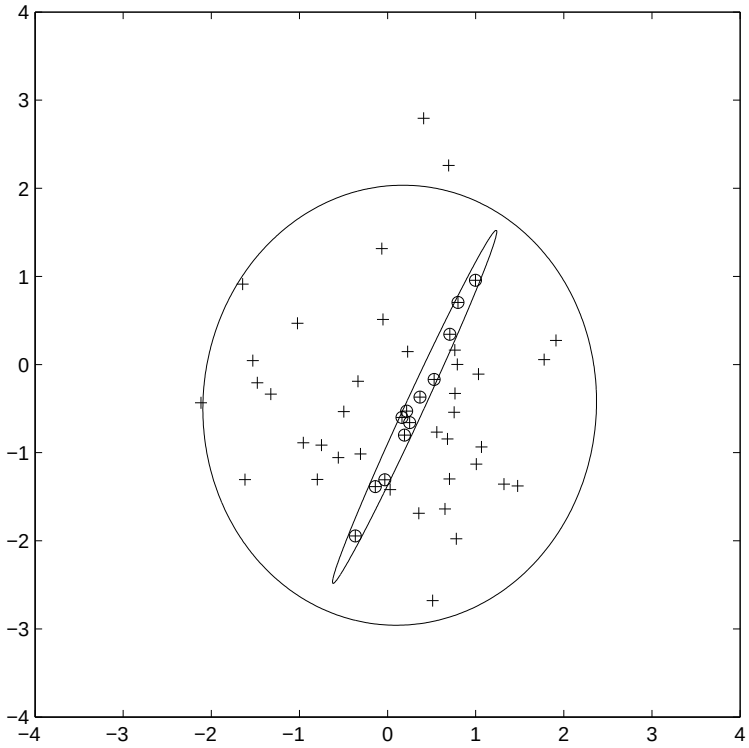


Figure 1.4: Data X_{50} with I_{12} points circled

identification and rejection followed by the use of standard estimates or tests." We would go further than this, and suggest that in a multivariate normal setting, correct identification will *always* be superior to (or at least as good as) robust estimation.* It is also true that the outliers themselves could be of interest, and are not necessarily nuisance observations produced by human error that need to be expelled from the data set and then forgotten.

One type of robust estimator attempts to reduce the influence of outliers by weighting the observations, with low weights to peripheral ones. An extreme form of this assigns a weight of zero to all suspicious observations and one for all others, which is equivalent to an ordinary estimate using an inlier subset. A well chosen inlier subset of sufficient size makes for an excellent robust estimator, and a plot of the Mahalanobis distances using such an estimate makes the visual identification of outliers as easy as in the univariate case.

*We are not undermining the field of robust estimation, only questioning its relevance to normally distributed data. It is useful in cases where outlying observations are not necessarily contaminants, so their effect is to be mitigated rather than eliminated [2] (p 3).

Problems do arise if the inlier subset is too small, even if the outliers are excluded. An example of this is shown in Figure 1.4 for a bivariate data set which we shall refer to as X_{50}. It consists of 50 observations from $N(\mathbf{0}, \mathbf{I})$. The sample estimates are again represented by 95% confidence ellipses, and in the case of I_{12} this is disastrous. The algorithm that found the inlier subset (in this case the MCD introduced in section 4.1) has discovered fortuitous structure within the cluster, which has lead to a non-representative subset and a degenerate estimate. A plot of the resulting Mahalanobis distances would be worthless.

1.6 Cluster analysis

Because we intend to deal with clusters as a type of (severe) contamination, a brief review of the discipline as it is commonly perceived is in order. The number of unique partitions of N objects into g groups is given by Stirling's number of the Second Kind*, which grows as $g^N/g!$, making a consideration of all of them impossible for all but the smallest samples and number of groups. A large number of heuristic algorithms have therefore been proposed, and they may be divided into the following two methods.

1.6.1 Hierarchical methods

A feature of these methods is that a *metric* is assumed to be known in advance. This is a distance measure that allows one to quantify the similarity or closeness of two points. The euclidean and Manhattan† distances are examples of a metric. A matrix is then constructed that contains all of the $\binom{N}{2}$ distances between the points, which serves as the input to the clustering algorithm. This has the advantage that no assumptions are made about the distribution of the clusters.

The groups are determined in stages, either by successive splitting of one or more groups and starting with the complete set, or by merging groups and starting with some initial partition. These are called *divisive* and *agglomerative* methods, respectively. At each stage, the operations carried out are chosen so as to optimize a similarity criterion based on the given metric, thus forming a hierarchy of partitions. The changes made at each stage are irreversible, so that if what are actually portions of two separate clusters are merged, then the final partition will be incor-

*See Appendix A.1 for details

†The distance travelled from one point to another when restricted to movement parallel to the Cartesian axes.

rect. Likewise, if a set of observations from a single cluster are split, then this error cannot be undone.

A drawback of the metrics used is that they are usually not *affine equivariant*, meaning that different answers may be obtained after a linear transformation of the data. While these may be suited to geometric (also known as spatial) clustering problems like the assignment of stars to galaxies, they are clearly undesirable for multivariate data with no geometric interpretation. Construction of the distance matrix also makes them $\Omega(N^2)$, which is fairly expensive. Kaufman and Rousseeuw [17] is a good introduction to hierarchical methods.

1.6.2 Non-hierarchical methods

These are also called *relocation* methods, because observations may be moved from one cluster to the next, in order to optimize some objective function. The best known of these is the K-means algorithm [15] (p 755). It is actually a special case of the more recent and general set of *model-based* clustering algorithms [1]. The clusters are usually assumed to be multivariate normal, with the objective function being the likelihood.

Two variations of model-based clustering are possible. They both view the data as being derived from a distribution composed of a weighted sum of normal densities. This is called a mixture distribution, and the proportions of each of the component distributions are called *mixing parameters*. Treating both the mixing parameters and the mean and variance of each component distribution as continuous variables over which the likelihood is to be optimized, is referred to as the *mixture* approach. Observations are subsequently assigned en masse to whichever component distribution gives them the largest likelihood. The *classification* approach uses a label for each observation that assigns it to a component distribution, in lieu of mixing parameters. A comparison of the two is given in [7], and for the special case of only two clusters, see [10].

Mixture models have been used for modelling outliers, with the contamination model usually having the same mean as the main distribution, but a larger variance. This is quite a restrictive assumption and is mostly useful as a conceptual tool [2] (p 46 ff). Mixture models are used extensively for density estimation (see [4], chap 2 for details), but in this application one is interested in describing the data rather analysing them, and a multi-component solution does not guarantee the presence of as many clusters.*

*For an extreme example of this, see [4] (p 68 ff), where a uniform, bivariate distribution over

1.6.3 Cluster analysis as an outlier problem

The likelihood methods of cluster analysis are widely used, but do have their limitations. It is difficult to allow for completely general parameters [1] (p 803), and restrictive assumptions are often made. For example, if all the covariance matrices are taken as being equal and proportional to $\mathbf{I}$, then we arrive at the K-means method.

Outliers are still a problem if the parameters of the clusters are not known *a priori* and have to be gleaned from the data itself. Each cluster must therefore contain at least $p+1$ observations for an estimate of its covariance matrix, so groups of outliers smaller than this will be problematic. The other difficulty is inherent to traditional cluster analysis, that of determining the number of clusters present [9] (p 512).

In a standard outlier detection procedure with κ unknown, it is often set to $\frac{1}{2}$, which is supposed to be the worst possible case. The argument is that contamination higher than this makes it impossible to distinguish between the inliers and outliers [25] (p 14). In a cluster analysis context, this is irrelevant. We therefore propose that an extension of the outward testing procedure for multiple outliers be used for cluster analysis. This identifies the clusters *individually* instead of *collectively* by allowing for $\frac{h}{N} < \frac{1}{2}$. All observations not belonging to the cluster currently being inspected are temporarily considered "contaminants", but κ itself still represents the maximum proportion of genuine contaminants that may exist.

Once a complete cluster is identified it is removed from the sample, and the process is repeated on the remaining data. Not only does this find the clusters, but also any outliers that might otherwise have gone unnoticed, or even caused incorrect categorization.

1.7 Outline

Chapters 2 and 3 serve to introduce essential concepts in a familiar univariate setting that are later applied to multivariate data. In the applied sciences, univariate normal data are more common than their more complex multivariate counterparts, so these may well be of interest in their own right.

Chapter 2 looks at ways of creating inlier subsets that are justified by likelihood arguments. We first consider combinatorial inlier subsets, and present the simple and

an annular region is approximated by a seven component Gaussian mixture model.

established algorithm for finding the optimal subset. We then see how sequentially generated inlier subsets compare by applying both techniques to a real-life data set.

Chapter 3 considers hypothesis testing, and begins with a detailed examination of a method of formalizing the testing of multiple outliers. We then cover the standard way of approximating significance probabilities using the Bonferroni inequality, and suggest a new alternative, which we call the Independence assumption. A simulation study investigates the accuracy of each method, and we conclude that although the Independence assumption is not quite as accurate as the Bonferroni inequality for low significances, it retains its reasonable accuracy over the entire domain of the test statistic. The usefulness of this property is illustrated by carrying out tests on the inlier subsets of the data set used in chapter 2.

Multivariate inlier subsets are the subject of Chapter 4. Of fundamental importance is the MCD, defined as the subset with minimum covariance determinant, and first proposed in 1984. Only in the last few years have algorithms been developed for finding this subset, but the usefulness of these is almost exclusively confined to bivariate data of not more than a few hundred observations. We describe and implement four of these exact algorithms.

Fortunately, a *heuristic* algorithm called Fast-MCD was also developed (shortly before the exact algorithms appeared), which finds what seem to be good solutions very quickly. Using known solutions found with the exact algorithms enabled us to conduct a thorough empirical analysis of Fast-MCD. This led to the discovery of an interesting convergence property that allows us to estimate the probability of it finding the bivariate MCD, irrespective of the sample size. We consider this to be a valuable original contribution to the field.

Fast-MCD is very effective for bivariate samples, but if the MCD is required with a high degree of certainty for large samples in high dimensions, then it takes a significant amount of time. We therefore propose a novel way of finding the inlier subsets, using a single MCD estimate of size h. We term this the *recovery* of subsets from the starting subset, and show that it is both effective and efficient.

Chapter 5 considers multivariate hypothesis testing, with most of what was said in chapter 2 applying equally well to the multivariate case. The results of further simulations for testing the same two methods of approximating significance probabilities as were used for univariate test statistics are presented. The Independence assumption is found to produce results that improve with dimension, whereas the accuracy of the Bonferroni inequality degrades with dimension. In fact, at the commonly used significance of 5%, the Independence assumption is more accurate for

$p \geq 3$. An example of outlier detection of another real-life data set is presented, where we arrive at a different conclusion to that of the authors of the book from which it is taken.

We then describe how outward testing may be used for finding clusters by working through a biometric data set. Finding the MCD of a data set composed of clusters can be very expensive, so recovery of subsets is essential for this to be feasible. Just as importance is the fact that the Independence assumption is reasonably accurate for all significance probabilities. This new type of cluster analysis may be used in place of more traditional methods if the clusters are multivariate normal. This being the case, it is very effective at identifying them, even if they overlap slightly, and any outliers present are easily identified.

We discuss the conclusions arrived at in chapter 6, and also point to a number of possibilities for further research.

Chapter 2

Univariate inlier subsets

2.1 Minimum Variance (MVAR)

The concept of likelihood, which is the basis of maximum likelihood estimation (MLE) of the parameters of a distribution, is a measure of how well a sample fits a distribution. Construction of the inlier subsets used in this thesis also uses this principle. If the parameters are unknown, but a likelihood value is required, then one could choose to use the MLE parameters and the resulting maximum likelihood. Consider this sample likelihood L calculated for a sample of size n from a normal distribution,

$$\begin{aligned} L &:= \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi}\hat{s}} \exp\left[-\frac{1}{2}\left(\frac{x_i - \bar{x}}{\hat{s}}\right)^2\right] \\ &= \frac{1}{(2\pi)^{\frac{n}{2}}\hat{s}^n} \exp\left[-\frac{1}{2}\left(\frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{\hat{s}^2}\right)\right] \\ &= \frac{1}{(2\pi)^{\frac{n}{2}}\hat{s}^n} e^{-\frac{n}{2}}. \end{aligned} \tag{2.1}$$

It is seen that the sample likelihood is inversely proportional to the sample variance. If we define I_n as the subset with the lowest sample variance, then we arrive at what Viljoen [27] calls the MVAR (minimum variance) method. Calculating the variance for all $\binom{N}{n}$ possible sample combinations would eventually produce the optimal one, but this is unnecessary, since the sample that produces the minimum variance must be a contiguous set from the ordered observations, which limits the solution to one of only $N - n + 1$ subsets.

To see this, suppose that the minimum variance is obtained from a non-contiguous

sample. Now add one of the internal but excluded observations to create a sample of size $n+1$. It will be seen in section 2.2 that when removing a single observation so as to minimize the variance of the remaining sample, one of the two extreme values must be discarded. Hence, a new sample of size n that includes the internal observation has been formed that has a variance less than the original sample of size n. This is a contradiction, so we conclude that the MVAR sample must be contiguous.

In formulating an algorithm for finding the MVAR subset based on this result, we refer to each contiguous sample as $C_j = \{x_{(j)}, \ldots, x_{(j+n-1)}\}$, with $j = 1, \ldots, N-n+1$. The variance and mean of C_j are denoted by $\hat{s}_j^2$ and $\bar{x}_j$. Their update formulae are easily verified as,

$$\hat{s}_{j+1}^2 = \hat{s}_j^2 + d_j \left[x_{(j)} + x_{(j+n)} - d_j - 2\bar{x}_j \right], \tag{2.2}$$

and

$$\bar{x}_{j+1} = \bar{x}_j + d_j, \tag{2.3}$$

where

$$d_j = \frac{x_{(j+n)} - x_{(j)}}{n}. \tag{2.4}$$

A simple search for the minimum variance based on these formulae is an efficient way of finding an MVAR sample, but we require the set of MVAR subsets $\{I_h, I_{h+1}, \ldots, I_N\}$, and are interested in the number of computations required to find them.

In keeping with standard practice in algorithm analysis, we count only division and multiplication operations, since addition and subtraction are relatively much faster. We first need to calculate the variance of the starting subset. This requires $h+2$ operations. Updating the variance requires only two operations, the multiplication of d_j and $[\cdot]$ in (2.2), and the division by n in (2.4) to obtain d_j. The multiplication by 2 in (2.2) can be compiled as an addition. This update formula is repeated $N-h$ times. For the next MVAR subset calculations, the new starting variance is determined from the previous (see (2.9)) in 4 operations. The total number of operations is therefore

$$h - 2 + \sum_{n=h}^{N-1} [4 + 2(N-n)] = N^2\kappa^2 + N(1+4\kappa) - 2, \tag{2.5}$$

where we have replaced h with $N(1-\kappa)$ for clarity. Because κ is normally assumed to be constant, the algorithm is $O(N^2)$. Strictly speaking, this is something of a

simplification, as is noted in [2] (p 132), where it is remarked that "it is reasonable to consider three potential outliers in a data set of 10 observations, but it is unrealistic to expect 30 outliers out of a data set of 100 observations." They suggest that the relationship of the form $k = \sqrt{N}$ be used, based only on intuition. A formal method, discussed by Hawkins [13] (p 61), assumes that each observation of the sample is, independent of the others, an outlier with probability π. The number of outliers in the sample is then a binomial random variable with parameters N and π. To be $(1 - \beta) \times 100\%$ certain that our chosen upper limit is not exceeded by the actual number of outliers present,

$$\sum_{i=0}^{k} \binom{N}{i} \pi^i (1-\pi)^{N-i} \geq 1 - \beta. \tag{2.6}$$

If the smallest k such that this is true is used, and if $\pi = 0.1$, $\beta = 0.05$, then a sample of size 10 produces $k = 3$. One of size 100 requires $k = 15$, so κ is seen to vary with N. Although this should be borne in mind, $\kappa \to \pi$ as $N \to \infty$, and in practice it is in any case common to be conservative and take $\kappa = \frac{1}{2}$.

The MVAR subset tends to find the group of n contiguous observations which is the most densely clustered. This is very likely to be near the centre of the distribution, regardless of the presence of outliers. The arithmetic mean of the MVAR sample is therefore an excellent robust estimate of the population mean. Indeed, the Least Trimmed Squares (LTS) estimator used in [25] was designed as a robust estimator of location in a univariate setting, and is equivalent to this method. This is not immediately obvious from its definition, which is given in a regression context [25] (p 15). It is defined as that parameter which minimizes the sum of the n smallest squared residuals. Notice that a parameter has to be decided upon before the squared residuals can be calculated, ordered, and the n smallest added. It is simply stated in [25] that the LTS estimator is the mean of the contiguous subset of size n of the ordered observations with least variance. Their algorithm [25] (pp 171-172) is the same as the MVAR algorithm just covered, but they do not prove that this must find the LTS estimator.

Our proof begins by choosing a value θ for the estimate of the mean. It is then clear that the n smallest residuals will be those of the n closest observations to θ, and that these form a contiguous subset C_j of size n. If θ is increased, then the residual of $x_{(j+n)}$ will decrease, and come into competition with that of $x_{(j)}$ for nth smallest residual. The new contiguous subset will eventually change to C_{j+1}, and the value of θ at which this occurs is of course the midpoint of $x_{(j)}$ and $x_{(j+n)}$. A

similar argument holds when θ is decreased, so the sum of the n smallest squared residuals for

$$\theta \in \left(\frac{x_{(j-1)} + x_{(j+n-1)}}{2}, \frac{x_{(j)} + x_{(j+n)}}{2} \right),$$

is therefore

$$\sum_{i=j}^{j+n-1} (x_{(i)} - \theta)^2.$$

This is a standard least-squares problem, with the sum of the residuals as a function of θ being parabolic. It is well known that the minimum occurs at $\theta = \bar{x}_j$, but this point need not fall within the interval under consideration. The expression for the sum of the residuals evaluated at the sample mean is equal to the MLE of the variance multiplied by n, and since n is constant, minimizing one also minimizes the other, so we use the terms interchangeably.

If the sample mean falls outside the interval, then the variance is a lower bound for the actual minimum over the interval. If the lowest of these lower bounds *does* occur inside the interval, then it must be the sought after global minimum. We therefore require that the sample mean of the contiguous subset with smallest variance (i.e. MVAR) falls within the interval. Suppose that $\hat{s}_j^2$ is the minimum variance. This implies that $\hat{s}_{j+1}^2 > \hat{s}_j^2$, so from (2.2) we see that

$$d_j \left[x_{(j)} + x_{(j+n)} - d_j - 2\bar{x}_j \right] > 0,$$

and since $d_j > 0$ this in turn implies that

$$\bar{x}_j < \frac{x_{(j)} + x_{(j+n)}}{2},$$

so the mean satisfies the upper bound of the interval. Re-indexing (2.2) to relate $\hat{s}_j^2$ and $\hat{s}_{j-1}^2$ yields

$$d_{j-1} \left[x_{(j-1)} + x_{(j-1+n)} - d_{j-1} - 2\bar{x}_{j-1} \right] < 0.$$

Re-indexing (2.3) shows that $\bar{x}_{j-1} = \bar{x}_j - d_{j-1}$, and by substituting this into the above inequality and using the fact the $d_{j-1} > 0$, it is seen that

$$\bar{x}_j > \frac{x_{(j-1)} + x_{(j-1+n)}}{2},$$

so the mean also satisfies the lower bound of the interval. Notice that for the valid

interval of C_1 and C_{N-n+1} there is no lower and upper bound, respectively, so this result holds for $j = 1, \ldots, N - n + 1$.

2.2 Extreme Deviate Removal (EDR)

The idea of sequentially removing deviant observations to produce a set of inlier subsets is now applied to the likelihood result of (2.1). Consider a sample of size n and the request that we are to remove the most extraneous observation. It might be instinctive to focus on the observation itself, but an alternative is to consider the remaining sample. Of the n possible subsets of size $n - 1$, that with the greatest likelihood could be chosen. At first sight, it would appear that one would have to calculate each of the n variances in order to find the minimum, but the algorithm actually turns out to be much simpler. Some straightforward algebra shows that

$$\hat{s}_d^2 = \frac{n}{n-1}\left(\hat{s}^2 - \frac{(x_d - \bar{x})^2}{n-1}\right), \tag{2.7}$$

where $\hat{s}_d^2$ is the variance of $\{I_n \backslash x_d\}$. It is clear that in order to minimize $\hat{s}_d^2$ the distance from the observation to the mean must be maximized. Since only the two most extreme observations need be considered, this is equivalent to removing $x_{(n)}$ if

$$\frac{x_{(n)} + x_{(1)}}{2} > \bar{x}, \tag{2.8}$$

and $x_{(1)}$ otherwise. Thus, a simple comparison followed by updating of the mean is all that is required to obtain I_{n-1} from I_n. It is interesting to note that this procedure is described by Viljoen [27] (chap 2, p 7) as "intuitively appealing but not based on a formal statistical method." We have shown that it does in fact have a theoretically sound basis, and although we can simply apply this step $k - 1$ times in order to obtain all the desired inlier subsets, this is not recommended, for reasons addressed in the next section.

2.3 EDR with recovery

Using EDR does not guarantee that all outliers will be removed before any inliers are affected. For instance, the data might consist of two well separated clusters with the same variance. The midpoint of the two extreme values could then oscillate about the mean for a number of removals, causing observations from both clusters to be

removed, before what is left of one of them dominates and the algorithm "latches onto" onto it. This is an instance of swamping.

A means of reclaiming those observations that may have been incorrectly discarded during EDR is needed. In order to achieve this, consider a subset of size n and add that observation x_a which minimizes the variance of the new sample of size $n+1$. This new variance is

$$\hat{s}_a^2 = \frac{n}{n+1}\left(\hat{s}^2 + \frac{(x_a - \bar{x})^2}{n+1}\right), \tag{2.9}$$

In order for $\hat{s}_a^2$ to be the minimum variance, it is seen that x_a must be the closest observation to the present sample mean. If we adopt the notation $x_{(0)}$ and $x_{(n+1)}$ for the closest observations to the subset, then it is obvious that one of these observations produces the required minimum variance. The analogue of (2.8) is that $x_{(0)}$ is added if

$$\frac{x_{(0)} + x_{(n+1)}}{2} > \bar{x}, \tag{2.10}$$

and $x_{(n+1)}$ otherwise. This is a fast way of finding I_{n+1} from I_n. We have to start somewhere, and we term the observations that make up the starting subset of this *recovery* procedure the *seed subset*. If this subset is indeed free of outliers, then recovery of observations will be very likely to proceed through all of the inliers before adding any outliers.

Two questions arise naturally from the previous discussion. How large should the seed subset be, and what technique should be used to find it? The EDR of the previous section could be applied to the sample until it is thought that the remaining observations are all inliers. To maximize the probability of this being the case, one could reduce the EDR subset to the minimum size needed for an estimate of the variance. EDR could therefore be applied until just two observations remain. Of course, one could also obtain a useful seed subset from an MVAR subset, in which case the seed subset size might as well be h. It will be seen that in the multivariate case, this alternative is mandatory.

Although MVAR subsets are optimal in the sense that all possible combinations are considered, the important difference between MVAR and the recovered subsets is that the latter produces all the required subsets in one sweep through the data. MVAR must be repeated for each, making the algorithm quadratic rather than linear. Both algorithms require that the data are sorted, which in general is $O(N \log N)$. MVAR is therefore still quadratic, but recovery becomes $O(N \log N)$.

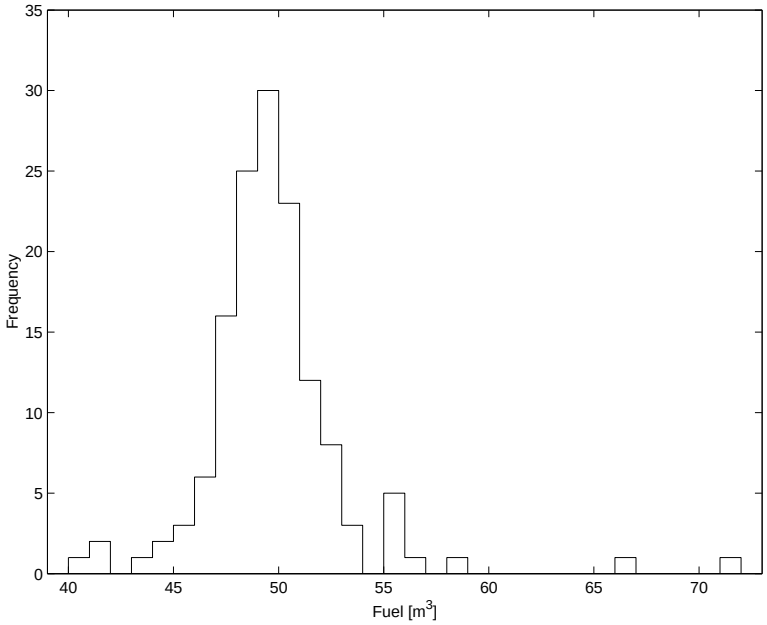


Figure 2.1: Histogram of ferry fuel data.

In the next section it will be seen that most of the MVAR and recovered subsets are identical. The use of a faster algorithm has resulted in very little sacrifice of performance. It is also worth mentioning that recovery is only necessary for univariate data when they are composed of clusters that might cause swamping in EDR. In most cases, EDR alone will perform adequately.

2.4 Example - Ferry fuel data

The data used in this example are from [18] (Table D.6, p 414), and are 141 records of the amount of fuel (in cubic metres) used by a ferry between the same ports. The histogram of these data shown in Figure 2.1 clearly reveals two upper outliers, whose origins are discussed in [18] (p 57). The author of [18] "was able to check the original records, taken from the ship's log, and found there had been gale force headwinds on those passages." The rest of the data appear to be normally distributed.

To illustrate the process of finding the inlier subsets, we have chosen to take $h = \lceil N/2 \rceil = 71$, as is commonly done when there is no information on the number of outliers present. A plot of the ordered observations is given in Figure 2.2, along with the range of the I_{71} and I_{100} MVAR subsets. Notice that they centre around

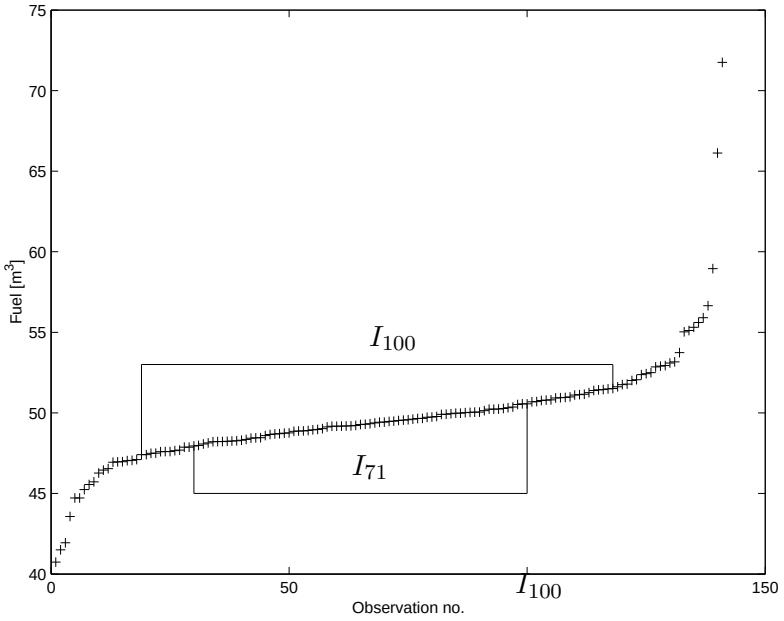


Figure 2.2: The ordered ferry fuel data showing two MVAR subsets.

the mode of the distribution, and effectively truncate the tails of the distribution as a result. This has important implications when multiple hypothesis testing is considered in section 3.1.

The recovered subsets are so similar to the MVAR subsets that a separate plot is unnecessary. Only 9 of the 71 subsets differ. Of these 9, 3 differ by only one shift of the starting indices, 2 by two shifts, and 4 by three shifts. Most importantly, the subsets from size 109 to 141 are all identical. It is in this region that outliers are most likely to be found, and differing subsets would result in different tests being conducted, and possibly different conclusions being drawn.

Chapter 3

Univariate hypothesis testing

3.1 Multiple outliers

Discordancy testing of a single outlying observation was presented in the introduction. The test statistic we described was

$$T_n := \max_{x_i \in I_n} \frac{|x_i - \bar{x}|}{s}, \tag{3.1}$$

which is known as the Extreme Studentized Deviate* or ESD. Notice that s is the square root of the *unbiased* estimate of the variance. A critical value c_n is calculated using the distribution of T_n under the null hypothesis H_n that exactly n inliers are present. That is,

$$\Pr\{T_n > c_n | H_n\} = \alpha. \tag{3.2}$$

The small probability α (typically 5%) of a Type I error is then well understood.

In the multiple outlier problem, Rosner [22] proposed that finding more outliers than are actually present should be defined as the Type I error. In the framework of a consecutive procedure, this implies that critical values $c_j : j = \{h, \ldots, N\}$ be found such that†

$$\Pr\left\{\bigcup_{j=h}^{n} T_j > c_j | H_n\right\} = \alpha, \quad \forall n = \{h, \ldots, N\}, \tag{3.3}$$

where T_j is the ESD of I_j. Rosner does not delve into how one would go about

*It is equivalent to test **N2** in [2] (p 223).

†Rosner's notation emphasizes the outliers, but we prefer the notation of Viljoen [27], which emphasizes the inliers.

finding these critical values, but we feel that a description reveals the complexity of (3.3). For an outward procedure, we start with $n = h$ and find c_h from the distribution of $T_h|H_h$ such that

$$\Pr\{T_h > c_h|H_h\} = \alpha. \tag{3.4}$$

To find the next critical value c_{h+1}, we see from (3.3) that

$$\Pr\{T_h > c_h \cup T_{h+1} > c_{h+1}|H_{h+1}\} = \alpha, \tag{3.5}$$

so the joint distribution of T_h and T_{h+1} under H_{h+1} is needed. Notice that in order to solve (3.5) for c_{h+1}, it has been tacitly assumed that

$$\Pr\{T_h > c_h|H_{h+1}\} < \alpha. \tag{3.6}$$

In general, finding c_n once the critical values before it are known, requires the joint distribution of the $T_j : j = \{h, \ldots, n\}$ under H_n, which makes for a very difficult problem. Even if this were known, the fact that sets of critical values would need to be calculated for all h, and all methods of inlier subset creation, would make their tabulation and use cumbersome.

A simplification of (3.3) is needed, and we begin by noting that assumption (3.6) is not at all unreasonable. Provided that h is large enough to produce predictable inlier subsets, I_h under H_{h+1} is likely to consist of the normal sample of size $h + 1$ minus its most suspicious or deviant observation. The distribution of the observations in I_h will have less prominent tails because of this trimming than a complete normal distribution, so that

$$\Pr\{T_h > c_h|H_{h+1}\} \ll \Pr\{T_h > c_h|H_h\} = \alpha, \tag{3.7}$$

and (3.6) is easily satisfied. Consequently, the probability of the second event in (3.5) will not be much lower than α. We can therefore argue that in the unlikely event that the most deviant observation in I_h causes T_h to exceed its critical value, then it is not unreasonable to suppose that the test statistic of I_{h+1}, which should contain the complete normal sample of size $h + 1$, will exceed *its* critical value too. That is,

$$\Pr\{T_{h+1} > c_{h+1}|T_h > c_h \text{ and } H_{h+1}\}, \tag{3.8}$$

is fairly high, which essentially means that there is considerable overlap of the two

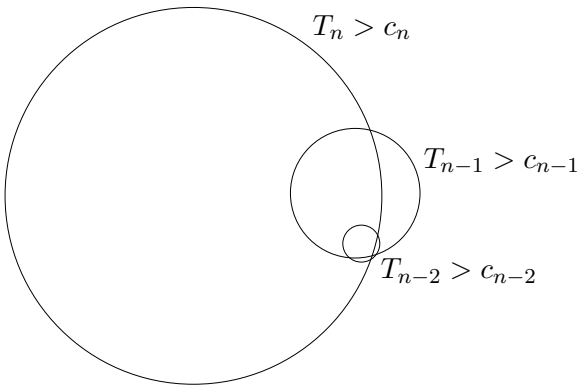


Figure 3.1: Venn diagram of the last three events of (3.3) under H_n

events in (3.5). This, combined with the smaller probability of the first event, provides reason to believe that

$$\Pr\{T_h > c_h \cap T_{h+1} < c_{h+1} | H_{h+1}\}, \tag{3.9}$$

is very small, and that as a result we can expect the solution of

$$\Pr\{T_{h+1} > c_{h+1} | H_{h+1}\} = \alpha, \tag{3.10}$$

to be very similar to that of (3.5). Notice that none of the assumptions made in arriving at (3.10) are contradicted by it. In fact, it reinforces the arguments leading up to (3.8).

We can extend the preceding arguments to the calculation of the remaining critical values by saying that in general

$$\Pr\{T_{n-1} > c_{n-1} | H_n\} \ll \Pr\{T_{n-1} > c_{n-1} | H_{n-1}\}, \tag{3.11}$$

and by applying the above repeatedly, we propose that

$$\lim_{n\to\infty} \Pr\{T_h > c_h | H_n\} = 0. \tag{3.12}$$

The general form of (3.8) becomes

$$\Pr\{T_n > c_n | T_{n-1} > c_{n-1} \text{ and } H_n\}, \tag{3.13}$$

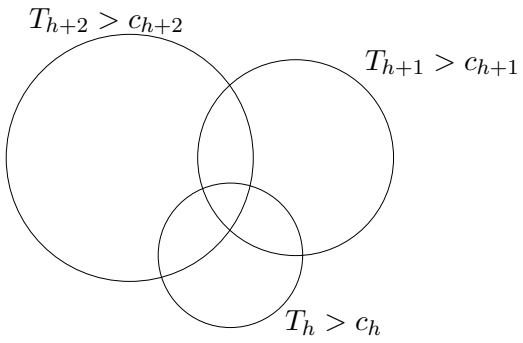


Figure 3.2: Venn diagram of the first three events in (3.3) under H_{h+2} for $N \leq 15$ and $\kappa \geq \frac{1}{3}$

and if this is again assumed to be high, then the limiting case becomes

$$\lim_{n\to\infty} \Pr\{T_n > c_n | T_h > c_h \text{ and } H_n\} = 1. \tag{3.14}$$

This limit is not really important, since we assumed in (3.12) that the probability of the event we are conditioning on tends to zero. These conjectures are summarized graphically for the last three events of (3.3) in the Venn diagram of Figure 3.1. We have attempted to construct it such that the proportions of the regions reflect the inequalities and limits put forward in equations (3.11)-(3.14).

We are now able to approximate (3.3) by

$$\Pr\{T_n > c_n | H_n\} = \alpha, \quad \forall n = \{h, \ldots, N\}, \tag{3.15}$$

which has reduced Rosner's criterion for the significance level of a multiple outlier test to nothing more than the application of the single outlier test applied sequentially to inlier subsets of increasing size. Rosner [22] simply states (3.15) as a conjecture, but we have provided some justification for it.

He goes on to conduct simulations using EDR inlier subsets at significances of 1% and 5%, in what is effectively an outward testing procedure. He chose to place the upper limit on the number of outliers as $k = \min(\lfloor N/2 \rfloor, 10)$, so for a sample of size 25 this results in $h = 16$. His approximation is vindicated for $N \geq 25$, but for $N \leq 15$ he found the true significance (as given by (3.3)), to be as much as twice the nominal α. We attribute this to the capricious nature of the smaller inlier subsets obtained from these samples when the proportion of contamination is large. Even if $h \geq 10$, a value of $\kappa \geq \frac{1}{3}$ means that the data are too sparse to produce reliable inlier

subsets. The assumption of (3.6) is probably still valid, but (3.7) and consequently (3.8) become tenuous. The Venn diagram for the first three events of (3.3) might look something like that of Figure 3.2. For $N \geq 25$, even $\kappa = \frac{1}{2}$ appears tolerable, and we see no reason to believe that any of the above assumptions will break down as $N \to \infty$.

3.2 Approximating the null distribution

3.2.1 The Bonferroni inequality

Rosner obtained approximate critical values for the ESD using the Bonferroni inequality, first used in this context by Wilks [29], who was interested in the more general multivariate case. The significance probability for T_n is

$$\begin{aligned}
\Pr\{T_n > c_n | H_n\} &= \Pr\{\max_{i \in I_n} |x_i - \bar{x}|/s > c_n | H_n\} \\
&= \Pr\{\bigcup_{i \in I_n} |x_i - \bar{x}|/s > c_n | H_n\} \\
&\leq \sum_{i \in I_n} \Pr\{|x_i - \bar{x}|/s > c_n | H_n\} \\
&= n \Pr\{|x_i - \bar{x}|/s > c_n | H_n\} \\
&= 2n \Pr\{(x_i - \bar{x})/s > c_n | H_n\}. \qquad (3.16)
\end{aligned}$$

The last step follows from the symmetry of the random variable $(x_i - \bar{x})/s$. The Bonferroni inequality sums the probabilities of the events separately instead of considering their union. If there is little overlap between them, then it is reasonable to assume that the two results will differ little. The smaller the significance probability the more rare it is for *any* of the Studentized deviates to exceed the critical value, and the chance of *more* than one being above the critical value becomes *extremely* unlikely. There is consequently less overlap of the events, and we can expect the approximation to improve with diminishing significance probability. If the probability given in the last line of (3.16) is $\frac{\alpha}{2n}$, then α is an upper bound for the actual significance.

It can be shown ([27] chap 2, p 18) that $(x_i - \bar{x})/s$ is a monotonic increasing transformation of a t_{n-2} random variable

$$\frac{x_i - \bar{x}}{s} \sim \frac{(n-1)t_{n-2}}{\sqrt{n(n-2+t_{n-2}^2)}}, \qquad (3.17)$$

allowing approximate critical values to be calculated from the extensively tabulated t distribution at significance $\frac{\alpha}{2n}$.

We prefer to work with the equivalent statistic

$$z_i := \frac{(x_i - \bar{x})^2}{(n-1)\hat{s}^2}, \tag{3.18}$$

where the relationship with T_n is

$$z_{(n)} = n\left(\frac{T_n}{n-1}\right)^2. \tag{3.19}$$

It obviates the need for taking absolute values, has the convenient domain of $(0, 1)$, and the z_i have the simple distribution* $\text{Beta}\left(\frac{1}{2}, \frac{n}{2} - 1\right)$, which easily generalizes to the multivariate case. If we let $f_\beta(z)$ denote its pdf, then the Bonferroni inequality produces

$$nf_\beta(z), \tag{3.20}$$

as an estimate of the pdf of $z_{(n)}$. This is in fact exact over a certain region, as noted by Wilks [29]. In order to show this, we notice from (3.18) that

$$\begin{aligned} \frac{n}{n-1} &= \sum_{i=1}^{n} z_i \\ &\geq z_{(n)} + z_{(n-1)} \\ &\geq 2z_{(n-1)} \end{aligned}$$

so that

$$z_{(n-1)} \leq \frac{n}{2(n-1)}. \tag{3.21}$$

If the critical value is greater than this upper bound on the second largest z_i, then the events in the second line of (3.16) are mutually exclusive, and all critical values greater than this upper bound are exact. Unfortunately, the significance probability of this upper bound, which is shown in the last column of Table 3.1, diminishes fairly rapidly with n. It is seen that the entire pdf for $n = 3$ is exact, and that exact critical values for $\alpha = 5\%$ are obtained for $n \leq 13$. Although not shown in Table 3.1, the exactness for $\alpha = 1\%$ extends to $n = 18$.

*A proof of this relationship between Student's t and the Beta distribution is provided in Appendix A.2

3.2.2 The Independence assumption

Because (3.20) integrates to n rather than one over $(0, 1)$, the Bonferroni inequality will become meaningless at the point where it exceeds one. We therefore introduce an alternative method of approximating the distribution of $T_n|H_n$, based on the assumption that the interdependence of the Studentized deviates is weak.

$$\begin{aligned}\Pr\{T_n > c_n|H_n\} &= \Pr\{\max_{i\in I_n} |x_i - \bar{x}|/s > c_n|H_n\} \\ &= 1 - \Pr\{\bigcap_{i\in I_n} |x_i - \bar{x}|/s < c_n|H_n\} \\ &\approx 1 - \prod_{i\in I_n} \Pr\{|x_i - \bar{x}|/s < c_n|H_n\} \\ &= 1 - [\Pr\{|x_i - \bar{x}|/s < c_n|H_n\}]^n. \end{aligned} \tag{3.22}$$

It is not unreasonable to expect this assumption of weak independence to become more accurate as $\bar{x}$ and s converge to μ and σ. This estimates the pdf of $z_{(n)}$ as

$$n\,[F_\beta(z)]^{n-1}\,f_\beta(z), \tag{3.23}$$

which is of course a true density function. Interestingly, the only difference between this and (3.20) is the factor $[F_\beta(z)]^{n-1}$, which will tend to one as z tends to one, so we can expect the two approximations to be asymptotically equal as z increases.

3.3 Simulation study

Extensive simulations were carried out to investigate the accuracy of the two approximations described above. For each of the n tested, 10^6 samples of standard normal random variables were generated and $z_{(n)}$ stored for each. This allows for very smooth histograms, such as the one in Figure 3.3. The vertical line at $z_{(6)} = 0.6$ demarcates the region over which (3.20) is exact. The other vertical line marks the exact 5% critical value, and it is clear that (3.23) is very close to the true pdf from this point on.

Just how close this match is can be seen from Table 3.1, where the true significance probabilities (estimated from the simulation data) for critical values* calculated at a nominal $\alpha = 5\%$ are presented for both methods. For the Independence ap-

*These are tabulated in [2] p 485, Table XIIIb. We choose to work with significance probabilities instead of the critical values themselves, for reasons given in section 3.4.

n	Bonferroni	Independence	$\Pr\left\{z_{(n)} > \frac{n}{2(n-1)}\right\}$
3	0.0495	0.0504	1
4	0.0499	0.0508	0.7340
5	0.0501	0.0511	0.5568
6	0.0498	0.0508	0.4229
7	0.0506	0.0517	0.3196
8	0.0500	0.0511	0.2402
9	0.0498	0.0510	0.1795
10	0.0498	0.0509	0.1335
11	0.0498	0.0510	0.0989
12	0.0498	0.0510	0.0730
13	0.0496	0.0509	0.0537
14	0.0497	0.0510	0.0394
15	0.0502	0.0514	0.0288
20	0.0499	0.0511	0.0059
50	0.0498	0.0510	0.0000
100	0.0494	0.0507	–
200	0.0496	0.0508	
500	0.0491	0.0504	
1000	0.0489	0.0502	

Table 3.1: Simulated significance probabilities for nominal $\alpha = 5\%$

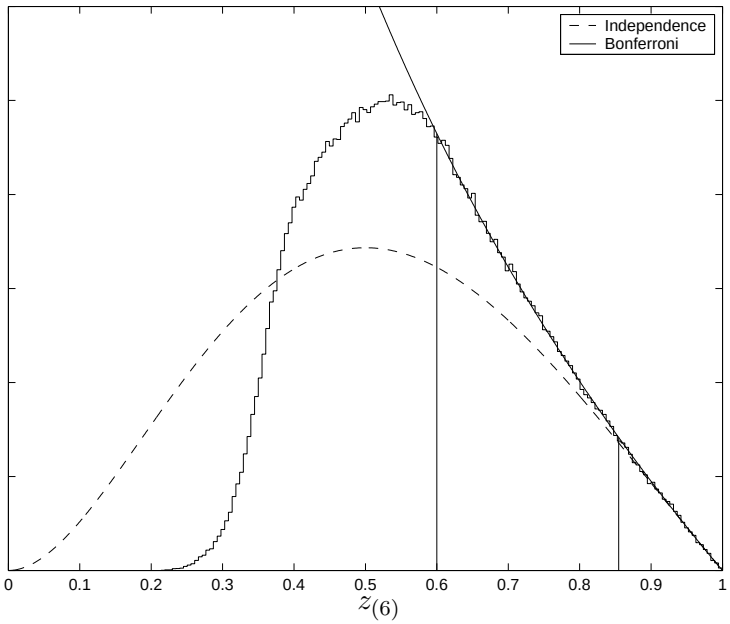


Figure 3.3: Histogram of simulation results for $n = 6$, with the two pdf approximations shown

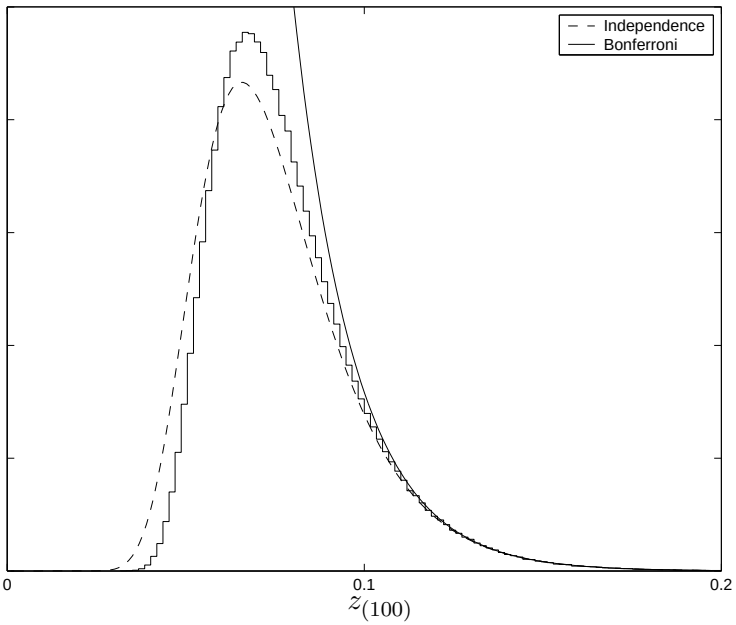


Figure 3.4: Histogram of simulation results for $n = 100$, with the two pdf approximations shown

proximation, these are consistently greater than 0.05, but certainly accurate enough to be useful. The exactness of the Bonferroni critical values for $n \leq 13$ provides us with a means of establishing the accuracy of the estimated significances. The average over this range of n is 0.0499, with a standard deviation from 0.05 of approximately 0.0003. For $n > 13$ the approximation seems excellent, and it is only for the largest two values of n simulated that a slight, but noticeable, drop below 0.05 is observed. On the other hand, the Independence approximation appears to *improve* for these large n, and it seems safe to say that it is more accurate than the Bonferroni inequality for $n > 500$. Since the approximation should converge to the true distribution as n increases, this is not surprising. The rate of this convergence is slow but steady. In Figure 3.3 the approximation is admittedly very crude, but the results for $n = 100$ in Figure 3.4 show a good overall fit of the histogram.

3.4 Example - Ferry fuel data

We now carry out hypothesis testing on the MVAR inlier subsets calculated in section 2.4. Significance probabilities of the test statistics $z_{(n)}$ were estimated using (3.23). These are plotted as a function of n in Figure 3.5. The first noteworthy feature of the plot is the very high significances for all the subsets smaller than about 110. These are derived from subsets consisting of the inner portion of the sample (See Figure 2.2), and clearly lie in the lower tail of their distributions. The probability of any of them being near the *upper* tail is therefore negligible, especially since the simulations of the previous section strongly suggest that (3.23) produces a lower bound for the true significance. That there is a remote probability of these test statistics exceeding their critical values is empirical evidence of (3.12).

The horizontal line marks the 5% cut-off significance, and I_{138} is the last inlier subset to pass the test. The result is that the three uppermost observations are declared outliers. Another common cut-off significance is 1%, and it is somewhat disconcerting that this concludes that only two upper outliers exist. There is no definite transition from clean inlier subsets to contaminated ones, with a cut-off of 10% finding five outliers.

Another curiosity is the steady increase from I_{129} to I_{134}. We would expect the significance probabilities to *drop rapidly* as the inlier subsets fill out into the tails of the sample. Anything else could signal that one cluster has been processed, and that the inlier subsets are beginning to encroach upon another. If the clusters overlap, then the first outlier to be included in the inlier subset might not be sufficiently dis-

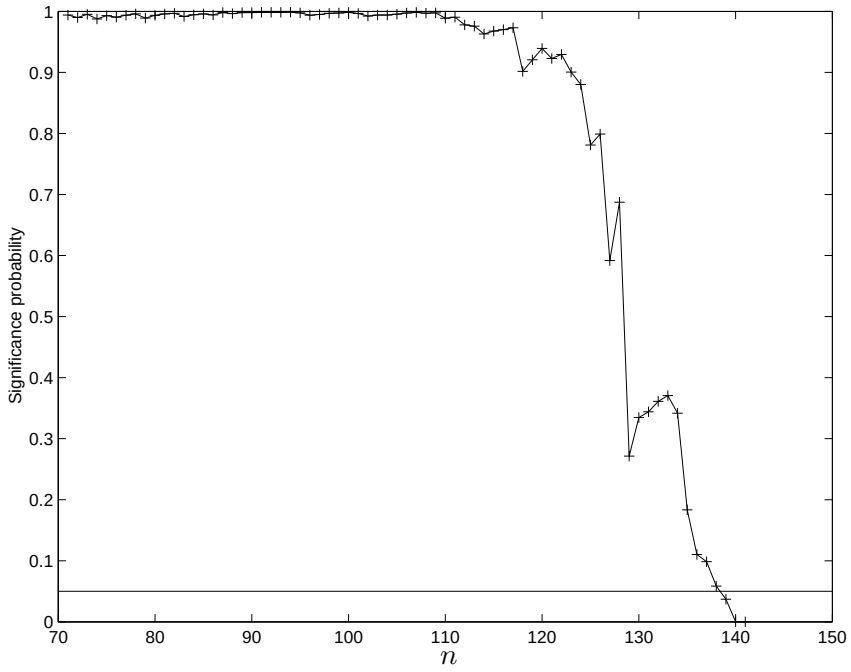


Figure 3.5: Ferry fuel data significance probabilities

cordant to arouse suspicion, and as more are included, the masking effect manifests in the form of stable or even increasing significances. In this example, we therefore conclude that the last acceptable inlier subset is I_{128}.

A histogram of the data is plotted in Figure 3.6, along with a suitably scaled density function that uses $\bar{x}$ and $\hat{s}$ derived from I_{138}. The same has been done for I_{128} in Figure 3.7. While the variance has clearly been overestimated by using I_{138}, that produced by I_{128} yields a strikingly good fit of the histogram. Two small clusters exist on both sides of the main distribution, along with the three upper outliers. This example illustrates the advantage of examining a plot of the significance probabilities rather than dogmatically comparing critical values. It also shows how useful it is to be able to estimate higher significance probabilities, and because this comes at very little cost to the accuracy of the smaller ones, we recommend the Independence approximation over the Bonferroni inequality.

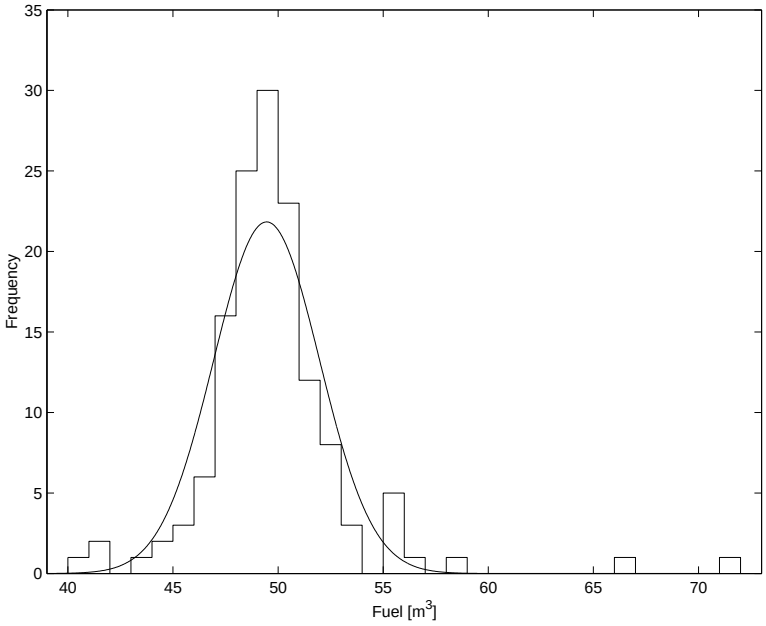


Figure 3.6: pdf derived from I_{138}

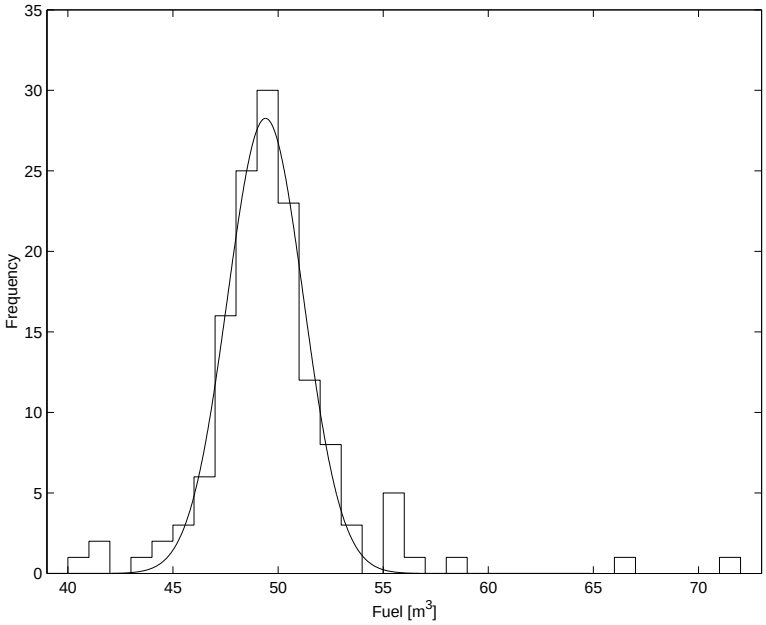


Figure 3.7: pdf derived from I_{128}

Chapter 4

Multivariate inlier subsets

4.1 The Minimum Covariance Determinant (MCD)

Many of the concepts from univariate data have multivariate counterparts. The likelihood principle that formed the foundation of the univariate techniques is no exception. In the multivariate case, we can apply the same sample likelihood principle used in (2.1) to obtain,

$$\begin{aligned} L &:= \prod_{i=1}^{n} \frac{1}{|2\pi\mathbf{S}|^{\frac{1}{2}}} \exp\left[-\frac{1}{2}(\mathbf{x}_i - \bar{\mathbf{x}})^T \mathbf{S}^{-1}(\mathbf{x}_i - \bar{\mathbf{x}})\right] \\ &= \frac{1}{|2\pi\mathbf{S}|^{\frac{n}{2}}} \exp\left[-\frac{1}{2}\sum_{i=1}^{n} \mathrm{tr}\{\mathbf{S}^{-1}(\mathbf{x}_i - \bar{\mathbf{x}})(\mathbf{x}_i - \bar{\mathbf{x}})^T\}\right] \\ &= \frac{1}{|2\pi\mathbf{S}|^{\frac{n}{2}}} \exp\left[-\frac{1}{2}\mathrm{tr}\{\mathbf{S}^{-1}\sum_{i=1}^{n}(\mathbf{x}_i - \bar{\mathbf{x}})(\mathbf{x}_i - \bar{\mathbf{x}})^T\}\right] \\ &= \frac{1}{|2\pi\mathbf{S}|^{\frac{n}{2}}} \exp\left[-\frac{1}{2}\mathrm{tr}\{\mathbf{S}^{-1}\mathbf{S}n\}\right] \\ &= \frac{1}{|2\pi\mathbf{S}|^{\frac{n}{2}}} e^{-\frac{pn}{2}}. \end{aligned} \tag{4.1}$$

It is seen that minimizing the determinant of the sample covariance matrix is equivalent to maximizing the sample likelihood L. This leads to the definition of the celebrated Minimum Covariance Determinant of Rousseeuw [23] (1984), who proposed that the subset with this lowest determinant be used for the robust estimation of the location and shape of a multivariate distribution. For ten years it remained a theoretical curiosity because of the overwhelming number $\binom{N}{n}$ of subsets to consider if an exhaustive search is conducted for anything but the smallest of samples. No

mention of it is made by Barnett and Lewis [2] (1994), but that same year Hawkins [14] introduced a *heuristic* algorithm that made the MCD practical for the first time. His pioneering algorithm has since been rendered obsolete by Rousseeuw and van Driessen [24] (1999), who developed a much faster heuristic algorithm called Fast-MCD.

Both of these heuristic algorithms apparently find good solutions, but only recently did Pesch [20] (2000), and Bernholt and Fischer [3] (2001) develop fast algorithms that are guaranteed to find the MCD. The following three sections review the algorithms of Pesch, after which we present the polynomial time algorithm of Bernholt and Fischer, which is the fastest exact algorithm we are aware of. We then investigate the properties of our implementations of all four exact algorithms.

Even Bernholt and Fischer's provably polynomial time algorithm is far too slow for samples containing more than a few hundred observations. The heuristic Fast-MCD is currently the only algorithm that is fast enough for use on large data sets, and is thus the only one used in practice. We describe it in detail in section 4.7 and go on to quantify its reliability in section 4.8 by using known solutions found with the exact algorithms.

4.2 The Branch and Bound Algorithm (MCD-BB)

4.2.1 Theory

An exhaustive search of all $\binom{N}{n}$ subsets may be implemented using the concept of a subset tree. Such a tree is shown in Figure 4.1 for the case $N = 6$ and $n = 3$. If one traverses along a branch to one of the leaves of the tree, then the subset is built up at each level until three unique and ordered numbers from the set have been passed along the way. These represent one of the $\binom{6}{3} = 20$ combinations.

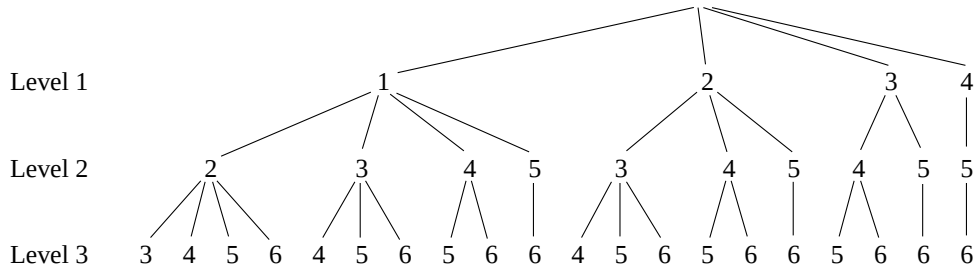


Figure 4.1: The subset tree

If at some level it becomes known that it is impossible for the subset currently being built up to be the MCD subset, then the branch from this node on may be ignored. In MCD-BB, this happens when a lower bound for the determinant of the subset exceeds the smallest determinant thus far found, and is described in section 4.2.2. A different termination condition is used in section 4.3. This technique is known as pruning the subset tree, and can dramatically reduce the number of nodes processed. To determine exactly how effective this pruning is, we need to compare the total number of nodes to the number actually visited during a given search. To calculate the total number of nodes, we first describe our implementation of the algorithm used to build the subset tree.

Consider an abacus wire with n beads, and a total of N discrete positions along the wire. The first bead's position (which represents the index of the first observation included in the subset) can vary from 1 to $N - n + 1$, allowing space for the rest of the $n - 1$ beads. The second bead can move from a position immediately to the right of the first, to $N - n + 2$, which leaves room for the remaining $n - 2$ beads. In general, the position of the lth bead (denoted by i_l) may vary from $i_{l-1} + 1$ to $N - n + l$. Positioning all n beads covers all $\binom{N}{n}$ possible combinations. Positioning only the first l beads is equivalent to covering all combinations on a wire with only

$N-l$ positions, so the number of ways of doing this (and thus the number of nodes on level l), is $\binom{N-n+l}{l}$. The reader may verify the preceding by counting the nodes on each of the three levels in Figure 4.1.

Using the identity

$$\binom{p}{r} = \binom{p+1}{r} - \binom{p}{r-1}, \tag{4.2}$$

we can calculate the total number of nodes as follows,

$$\begin{aligned}
\sum_{l=1}^{n} \binom{N-n+l}{l} &= \sum_{l=1}^{n} \binom{N-n+1+l}{l} - \sum_{l=1}^{n} \binom{N-n+l}{l-1} \\
&= \binom{N-n+1+n}{n} + \sum_{l=1}^{n-1} \binom{N-n+1+l}{l} - \sum_{j=0}^{n-1} \binom{N-n+1+j}{j} \\
&= \binom{N+1}{n} + \sum_{l=1}^{n-1} \binom{N-n+1+l}{l} - \sum_{j=1}^{n-1} \binom{N-n+1+j}{j} - \binom{N-n+1}{0} \\
&= \binom{N+1}{n} - 1.
\end{aligned} \tag{4.3}$$

which may also be checked using Figure 4.1.

4.2.2 Description

MCD-BB prunes the subset tree by calculating lower bounds on the determinant of the matrix $\mathbf{B}_n = n\mathbf{S}$. Since $|\mathbf{B}_n| = n^p|\mathbf{S}|$, minimizing the determinant of this matrix is equivalent to finding the MCD. We now consider how this matrix may be constructed by moving along a branch of the subset tree. We denote the index of the observation added on level l by i_l, and the sum of the l observations by $\mathbf{T}_l$. The required update formulae are

$$\mathbf{T}_{l+1} = \mathbf{T}_l + \mathbf{x}_{i_{l+1}}, \tag{4.4}$$

and

$$\mathbf{B}_{l+1} = \mathbf{B}_l + \frac{1}{l(l+1)} \left(l\mathbf{x}_{i_{l+1}} - \mathbf{T}_l\right) \left(l\mathbf{x}_{i_{l+1}} - \mathbf{T}_l\right)^T. \tag{4.5}$$

The first equation is immediate, and the second may be derived using basic matrix algebra. Taking the determinant of (4.5) gives

$$|\mathbf{B}_{l+1}| = |\mathbf{B}_l| \left[1 + \frac{1}{l(l+1)} \left(l\mathbf{x}_{i_{l+1}} - \mathbf{T}_l\right)^T \mathbf{B}_l{}^{-1} \left(l\mathbf{x}_{i_{l+1}} - \mathbf{T}_l\right)\right]. \tag{4.6}$$

Since $\mathbf{B}_l^{-1}$ is positive definite, the quadratic form in (4.6) is positive, so that $|\mathbf{B}_{l+1}| \geq |\mathbf{B}_l|$. Each $|\mathbf{B}_l|$ is thus a lower bound for the eventual $|\mathbf{B}_n|$, so if the determinant at some level $l < n$ is greater than the smallest $|\mathbf{B}_n|$ thus far found, then the current branch is aborted. The pseudocode is shown in Figure 4.2, and includes the code that constructs the subset tree.

The algorithm can be slightly improved by using a heuristic algorithm (See section 4.7) to produce a good estimate for the minimum $|\mathbf{B}_n|$, so that effective pruning takes place from the outset, instead of waiting for the algorithm to produce a low $|\mathbf{B}_n|$ itself.

4.2.3 Comments

When calculating MCD_{25} of X_{50}, the standard MCD-BB algorithm pruned about 99.99933% of the nodes. This increased marginally to 99.99936% when the actual minimum $|\mathbf{B}_n|$ was supplied beforehand. Pruning is clearly very effective, but these high percentages are misleading, since a large number (around 1.59×10^9) of nodes still need to be visited in the latter case.

A noteworthy feature of the pseudocode of Figure 4.2 is the while loop that ensures that level $p+1$ is reached before the main while loop begins. This is necessary for a non-singular covariance matrix, and entails visiting a certain minimum number of nodes, given by

$$\sum_{l=1}^{p+1} \binom{N-n+l}{l} = \sum_{l=1}^{p+1} \binom{(N-n+p+1)-(p+1)+l}{l} = \binom{N-n+p+2}{p+1} - 1, \tag{4.7}$$

where (4.3) is used to evaluate the summation. If we make the reasonable assumption that n is a linear function of N (see section 2.1), then this shows that the algorithm is $\Omega(N^{p+1})$, although in section 4.6.2 it is seen that the actual running time is unfortunately exponential.

4.3 The Sweepline algorithm (MCD-SW)

4.3.1 Theory

In the univariate case, it was seen that there exists a fast and simple algorithm for determining the MVAR subset that was based on the fact that it must be contiguous. One multivariate analogue* is that the *convex hull* formed by the MCD subset must

*Another analogue exists that is used in section 4.5

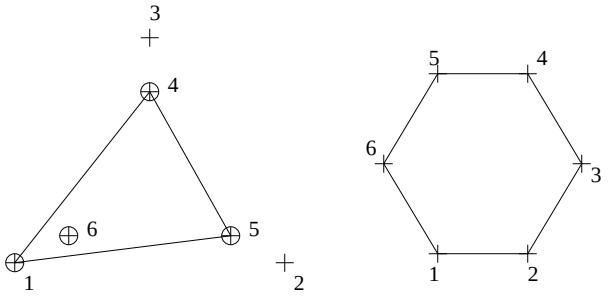
```
i := {1, 2, ..., n}
Add first point i_1, l := 1
do
   while l < p + 1
      Add point i_{l+1}, l := l + 1
   endwhile
   while TRUE
      if |B_l| > lowest |B_n| known
         l := l − 1
         break
      endif
      if l = n
         Store |B_n| and the subset
         l := l − 1
         break
      endif
      Add point i_{l+1}, l := l + 1
   endwhile
until next_subset returns FALSE

next_subset:
   i_{l+1} := i_{l+1} + 1
   while i_{l+1} > N − n + l + 1
      if l = 0
         return FALSE
      endif
      l := l − 1
      i_{l+1} := i_{l+1} + 1
   endwhile
   for j := l + 2 to n
       i_j := i_{j−1} + 1
   next j
   if l = 0
      Add first point i_1, l := 1
   else
      Add point i_{l+1}, l := l + 1
   endif
return TRUE
```

Figure 4.2: Pseudocode for MCD-BB (Source code in Appendix B.2)

contain only points that are part of this subset. We term such a subset *convex complete*. Although this is true for any dimension, we limit ourselves (as did Pesch) to bivariate data.

A simple example of such a subset, along with its convex hull, is shown in Figure 4.3 (a). The subset consists of the four circled points, with three *extreme points* defining the convex hull, and the fourth internal to it. Unfortunately, there is no simple way of finding all the convex complete subsets, nor even of determining how many exist in a given sample. In Figure 4.3 (a), the subset $\{2, 3, 4, 5\}$ is also convex complete. The reader may verify that 6 out of the total of $\binom{6}{4} = 15$ subsets are convex complete. The opposite extreme is shown in Figure 4.3 (b), where because all points of the sample lie on its convex hull, all subsets are convex complete. This is known as a *strongly convex* set.



(a) (b)

Figure 4.3: Two examples of bivariate data consisting of 6 observations

In practice it is found that the proportion of convex complete subsets is very small. The convex hull of the MCD_{25} subset of X_{50} is shown in Figure 4.4, where only 198823 of the total $\binom{50}{25}$ subsets (roughly $1\frac{1}{2}$ for every billion) are convex complete. The algorithm of Pesch [20] exploits this fact to produce two fast algorithms for finding the MCD subset, one of which is the subject of this section.

4.3.2 Description

Like MCD-BB, MCD-SW prunes the subset tree. Unlike MCD-BB, the data are first *lexicographically sorted*. This simply means that the points are sorted relative to one of the two variables. Our implementation assumes that the first variable (i.e. the first row of the $2 \times N$ data matrix) is used, and that this variable represents the x-coordinate on a scatter plot. That is, $x_i < x_j \Leftrightarrow i < j$, so moving down a branch

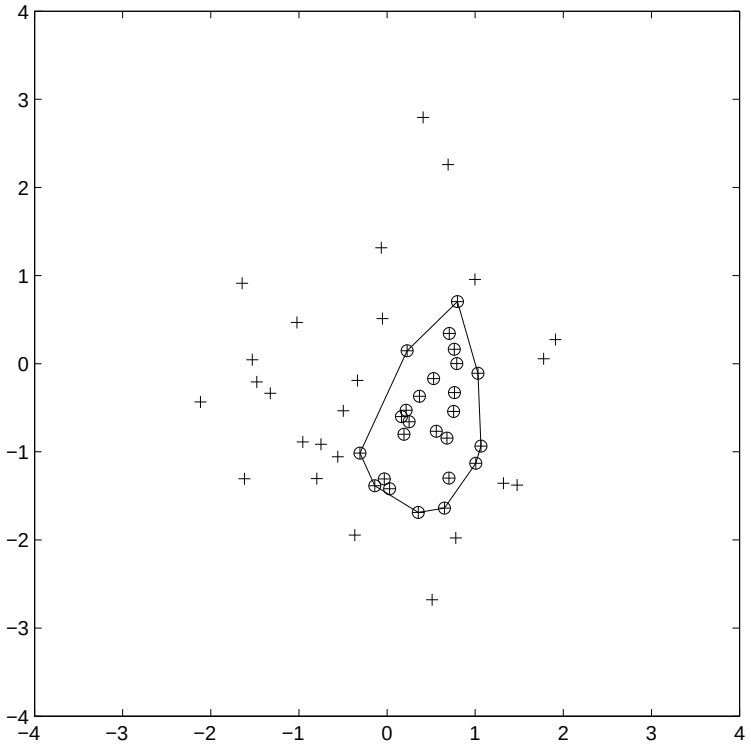


Figure 4.4: Convex hull of MCD_{25} of X_{50}

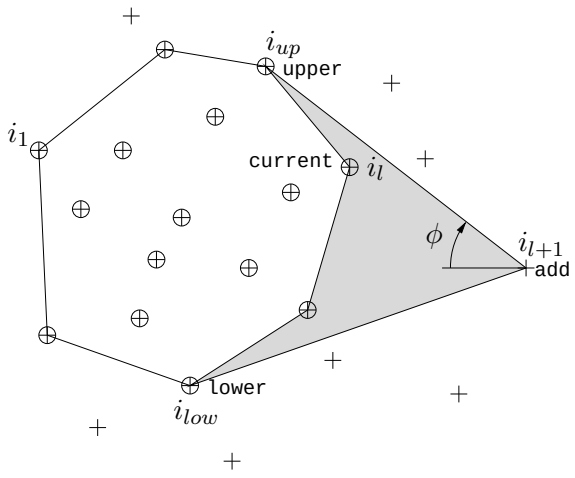


Figure 4.5: Incremental construction of convex hull

of the subset tree causes the subset to be built up from left to right.

The convex hull of the partial subset is updated at each level, as shown in Figure 4.5. If the new region thus added (show in grey) contains one or more points, then the final subset cannot be convex complete, and branch is terminated. This follows because the convex hull can only grow as new points are added, and because these lie to the right of `add`, it is not possible that foreign point(s) will ever be part of the subset.

The construction of the subset continues until either a foreign point is included in the new region, or until the subset is complete, in which case the determinant of its covariance matrix is calculated. Following this, a new subset must be set up.

4.3.3 Comments

The implementation of MCD-BB followed directly from its description. The geometric nature of MCD-SW however, means that a number of possibilities exist for its implementation. We refer the reader unfamiliar with geometric algorithms to [8], which we found was a useful primer for the subject.

Calculating the new region in Figure 4.5 entails finding the upper and lower tangents from the point being added to the existing convex hull. For the upper tangent, we achieve this by traversing along the convex hull counter-clockwise from the `current` point (added on the previous level) until the angle ϕ is smaller than at the previous point. The lower tangent is found similarly. The convex hull is stored

```
i := {1, 2, ..., n}
Add first point i_1
Add second point i_2, l := 2
do
   do
      check_point_and_add
   until it returns INVALID or COMPLETE
   if l = n
      Calculate |S|
      if |S| is smallest found
         Store it and the subset
      endif
      Restore convex hull to previous level, l := l - 1
   endif
until next_subset returns FALSE

check_point_and_add:
   Calculate tangents and search new region
   if New point is invalid
      return INVALID
   endif
   Construct new convex hull, l := l + 1
   if l = n
      return COMPLETE
   endif
return OK

next_subset:
   i_{l+1} := i_{l+1} + 1
   while i_{l+1} > N - n + l + 1
      if l = 0
         return FALSE
      endif
      Restore convex hull to previous level, l := l - 1
      i_{l+1} := i_{l+1} + 1
   endwhile
   for j := l + 2 to n
       i_j := i_{j-1} + 1
   next j
   if l = 0
      Add first point i_1
      Add second point i_2, l := 2
   elseif l = 1
      Add point i_2, l := 2
   endif
return TRUE
```

Figure 4.6: Pseudocode for MCD-SW (Source code in Appendix B.3)

as a double linked list to facilitate traversal in both directions.

When inspecting this new region, only those points from $x_{\min\{i_{up},i_{low}\}+1}$ to $x_{i_{l+1}-1}$ need be examined. A check is first made to see if one of these points lies in the triangle formed by `upper`, `lower` and `add`. If so, then a semaphore vector is consulted to see if it is already part of the sample. The function `check_point_and_add` in the pseudocode of Figure 4.6 performs these operations.

Each time a point is added and the convex hull changed, pointers to the start and end of the discarded segment (which consists of two points in Figure 4.5), along with the `current` point, are stored in the data structure `add`, to allow for fast restoration of the previous convex hull.

MCD-SW turns out to be a dramatic improvement over MCD-BB, with the former visiting only about 10.8×10^6 nodes. This is an improvement of more than two orders of magnitude over the latter (see 4.2.3). We also note that the extra effort of having to sort the data lexicographically is minuscule in comparison to this gain.

4.4 The Extreme algorithm (MCD-EXT)

4.4.1 Theory

A convex complete subset can be uniquely identified by the extreme points that form its convex hull. In most cases, these extreme points are only a small fraction of the total subset, so an algorithm that aims to process them is potentially much faster than one which deals with all of the subset points directly. MCD-EXT is such an algorithm, and it turns out to be the fastest, and most complex of Pesch's algorithms. Its complexity stems from the fact that the number of extreme points of a convex complete subset of size n may vary from 3 to n, so it cannot make use of a subset tree. So whereas MCD-BB and MCD-SW used the concept of a *level* to indicate the number of points included at any stage of the algorithm, we use the notion of *depth* in MCD-EXT to designate the number of extreme points included in the current subset. As in MCD-SW, the data are assumed to be lexicographically sorted, so points are added from left to right.

To understand the principle upon which MCD-EXT is based, consider the set of extreme points of a convex complete subset of size n and remove the rightmost of these. Figure 4.7 depicts an example configuration of the remaining extreme points, along with their convex hull. We now attempt to reconstruct the original convex complete subset by adding a valid extreme point. This must not displace the others,

since they are also extreme points of the final subset. The potential extreme point may therefore only "see" one edge of the present convex hull.

We are adding a point right of `middle`, so it will always be able to see at least one of edges `upper-middle` and `middle-lower`, so only these two need be considered. All points that can only see edge `upper-middle` and lie right of `middle` fall in the region $E_{u,d}$, which stands for the potential extreme points in the upper region for the present depth d. We could choose to add the first point in this region if it contains more than one, or take one from $E_{l,d}$ if it is empty. Once it is added, the inclusion of new internal points must be established. Assuming a point from $E_{u,d}$ was added, the potential internal points fall in the region $\{I_{u,d}, E_{u,d}\}$. The total number of points in the subset is now calculated, and if this equals n, then we have either found the original convex complete subset, or a new one.

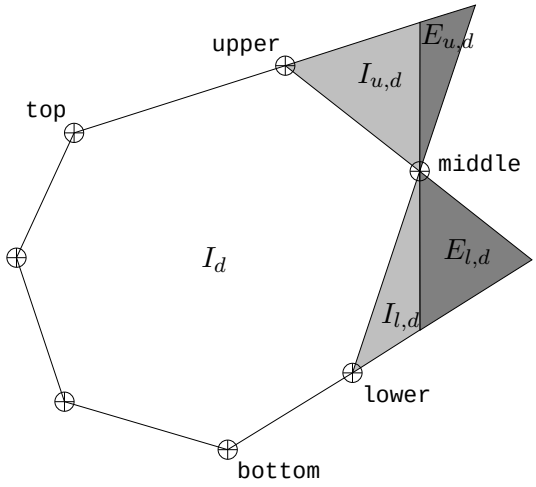


Figure 4.7: The five regions and points for depth d in MCD-EXT

4.4.2 Description

MCD-EXT uses the procedure explained in the previous section to find all the convex complete subsets of size n in a sample. This is the essence of the algorithm, and as with MCD-SW, there is plenty of scope for programming it. What follows is a description of our implementation of MCD-EXT.

To initiate the search, we set up the first two points manually by means of two nested for loops, as seen from the pseudocode in Figure 4.9. The index of the first point starts at 1 and ends at $N - n + 1$, which leaves just enough points right of it

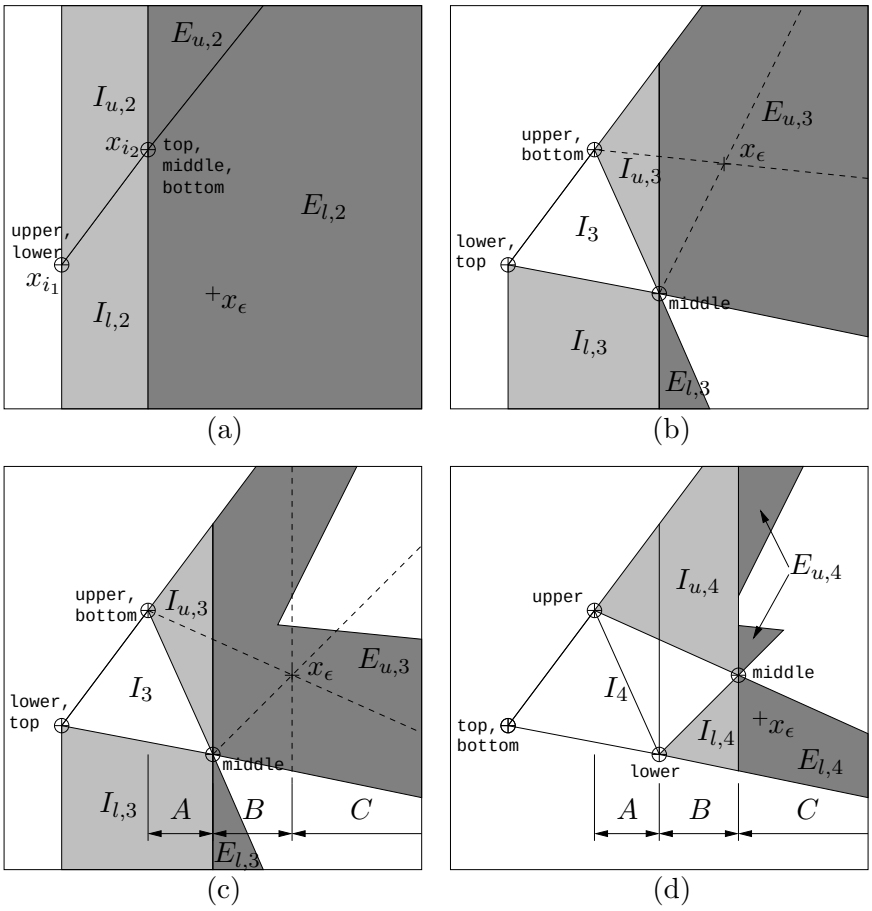


Figure 4.8: The first four depths of MCD-EXT building up a subset

to produce a subset of size n. The second index starts immediately to the right of the first, and continues until $N-1$, leaving just one point to form a triangle with the first two and thus stand a chance of producing a convex complete subset of size n. Once the two points are chosen, they are labelled appropriately and the regions, stored as linked lists, are calculated. The state is now that shown in Figure 4.8 (a). For as long as the depth is greater than or equal to two, the following general routine is repeated.

The subroutine `add_point` considers the addition of a new point x_ϵ by calculating the regions for the next depth. If x_ϵ lies in the upper extreme region, then $I_{u,d}$ and $E_{u,d}$ are scanned, since the new regions will be some subset of them. This property means that the maximum number of points that could be added to the subset is $|I_{u,d}| + |E_{u,d}|$. Thus if $|I_{u,d+1}| + |E_{u,d+1}|$ is less than the deficit for the next depth,

```
for i_1 = 1 to N − n + 1
   add_first_point
   for i_2 = i_1 + 1 to N − 1
      if add_second_point returns ERROR
         continue
      endif
      do
         do add_point while it returns OK
         if it returned DONE
            Calculate |S|
            if |S| is smallest found
               Store it and the subset
            endif
         endif
      while get_epsilon returns OK
   next i_2
next i_1

add_point:
   while processing ranges A and B
      if |I_{d+1}| + d = n
         Call update_E
         return ERROR
      endif
   endwhile
   I_{d+1} := {I_{d+1}, x_ε}
   if |I_{d+1}| + d + 1 = n
      Call update_E
      return DONE
   endif
   Process range C
   if |I_{u,d+1}| + |E_{u,d+1}| < n − |I_{d+1}| − d − 1
      Delete I_{u,d+1} and E_{u,d+1}
   endif
   (Repeat the above test for the lower regions too.)
   if {E_{u,d+1}, E_{l,d+1}} = φ
      return ERROR
   endif
   d := d + 1
return OK
```

Figure 4.9: Pseudocode for core routines of MCD-EXT (Pseudocode for supporting subroutines shown in Figure 4.10, with source code in Appendix B.4)

then these regions are deleted. Likewise for the lower regions.

The scanning is done in stages over the three domains labelled A, B, and C in Figure 4.8 (c) and (d), where a transition from depth 3 to 4 is shown. Two errors can occur that prevent the addition of x_ϵ. The first is the inclusion of more than n points in the subset. This can be established during the scanning of domains A and B, since no more points are added to I_d over domain C. The routine `update_E`

```
add_second_point:
   if |I_{u,2}| + |E_{u,2}| < n − 2
      Delete I_{u,2} and E_{u,2}
   endif
   (Repeat the above test for the lower regions too.)
   if {E_{u,2}, E_{l,2}} = φ
      return ERROR
   endif
   d := 2
return OK

update_E:
   Remove the potential extreme points beyond x_ε
   that cannot be included from this depth on.
return

get_epsilon:
   while no new points in {E_{u,d}, E_{l,d}}
      d := d − 1
      if d = 1
         return DONE
      endif
   endwhile
   set x_ε to the new extreme point
return OK
```

Figure 4.10: Pseudocode for supporting subroutines of MCD-EXT (Pseudocode for core routines shown in Figure 4.9, with source code in Appendix B.4)

is then called to delete the region beyond x_ϵ that is enclosed by the dashed lines in Figure 4.8 (b). Any extreme point in this region from this depth onward would add all those internal points that x_ϵ did (and possibly more), so it would also have produced too large a subset. This reduction of the future extreme regions can result in a modest improvement over Pesch's original algorithm. The next x_ϵ is then chosen from the remaining extreme region, as seen in Figure 4.8 (d).

The second error is when the algorithm runs out of extreme points to add before a subset of size n is obtained. If no error occurs, and the subset size is not n, then x_ϵ is added and the depth is increased. This process continues until either a subset of size n is found, or an error occurs. If the former, then the covariance determinant is calculated, and if it is the smallest found thus far, then it and the subset are stored. The state is subsequently returned to that of the previous depth by calling the function `get_epsilon`. This finds a new x_ϵ, backtracking through previous depths if necessary. If an error occurred, then it is called immediately.

4.4.3 Comments

It turns out that use of our routine `update_E` has little effect on the running time for half sample MCD subsets, which we use for testing purposes in section 4.6.2. The routine is only called (see Figure 4.9) when the subset contains n points, so the sooner this happens, the more points it will remove from the future extreme region. A considerable improvement *is* therefore realized for smaller MCD subsets. Finding quarter sample MCD subsets for a sample size of 80 was on average 15% faster when using the routine. On the other hand, three quarter subsets for the same sample size are *slower* by around 10%, so the overhead of the function call negates any advantage gained by removing points. The same is true of the half sample MCD, though it is only 2% slower. The routine is therefore excluded by default, but this may be changed by defining the macro `UPDATE_E` and recompiling.

4.5 Bernholt and Fischer's algorithm (MCD-BF)

4.5.1 Theory

It was mentioned in section 4.3 that there is a more specific condition than the subset being convex complete exists that must hold for the MCD subset. This is that the subset must be selectable by an ellipsoid, which in the bivariate case is an ellipse. By this we mean that an ellipse exists that contains only subset points in its interior region and one on its boundary. Because this region is convex, it is clear that the subset is also convex complete, since the convex hull is by definition the boundary of the smallest convex region containing the subset points, and therefore lies within the ellipse. The converse is not true. That is, a convex complete subset is not necessarily selectable by an ellipse. This suggests that an algorithm that searches for all subsets selectable by ellipses could be faster than those of Pesch.

If the MCD subset is known, then finding an ellipse that selects it is simple, since the locus of points with constant Mahalanobis squared distance is an ellipse. The maximum Mahalanobis distance of the subset (which is used in chapter 5 as a test statistic) is $z_{(n)}$, so

$$\tfrac{1}{n-1}(\mathbf{x}-\bar{\mathbf{x}})^T\mathbf{S}^{-1}(\mathbf{x}-\bar{\mathbf{x}}) = z_{(n)}, \tag{4.8}$$

selects the subset, as is shown in section 4.7. To clarify the nature of this quadratic

form, we multiply it out and consider the number and degree of the resulting terms.

$$\underbrace{\mathbf{x}^T\mathbf{S}^{-1}\mathbf{x}}_{\frac{p}{2}(p+1)\text{ quadratic terms}} - \underbrace{2\bar{\mathbf{x}}^T\mathbf{S}^{-1}\mathbf{x}}_{p\text{ linear terms}} + \underbrace{\bar{\mathbf{x}}^T\mathbf{S}^{-1}\bar{\mathbf{x}} - z_{(n)}(n-1)}_{1\text{ constant term}} = 0. \tag{4.9}$$

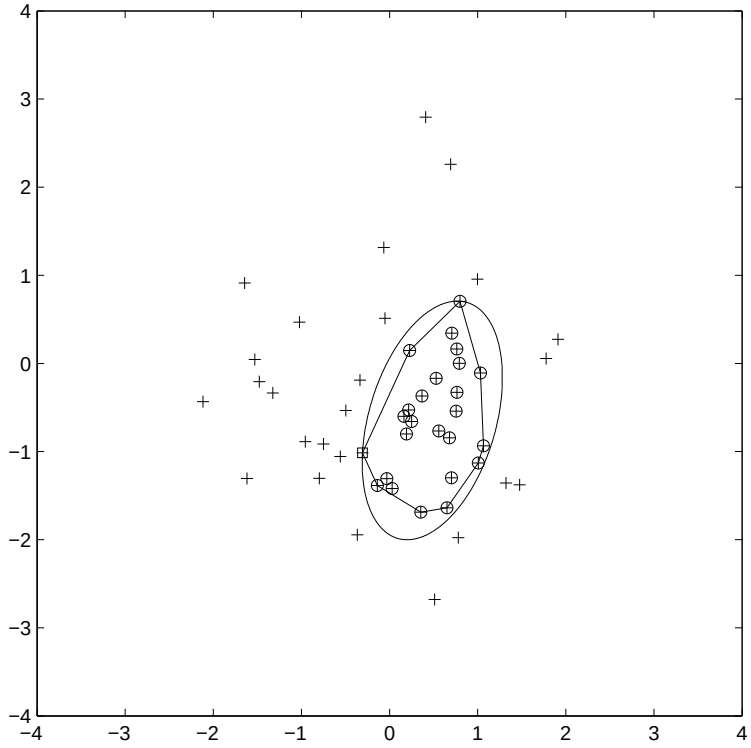


Figure 4.11: The ellipse of (4.8) selecting the MCD_{25} subset of X_{50} with convex hull also shown.

The original expression is negative if evaluated for some $\mathbf{x}$ internal to the ellipsoid, zero if it is on the boundary, and positive if it is external. Although multiplying it by a constant doesn't change it (and it therefore has $\frac{p(p+3)}{2}$ degrees of freedom), we do require that this constant be positive so as not to invert the selection. In the bivariate case it may be written in standard form as

$$Ax^2 + Bxy + Cy^2 + Dx + Ey + F = 0, \tag{4.10}$$

and because $\mathbf{S}^{-1}$ is positive definite, we know that both A and C are positive. We may therefore assume, without loss of generality, that $A = 1$. Now consider what happens if we gradually vary the remaining coefficients, but with the constraint that

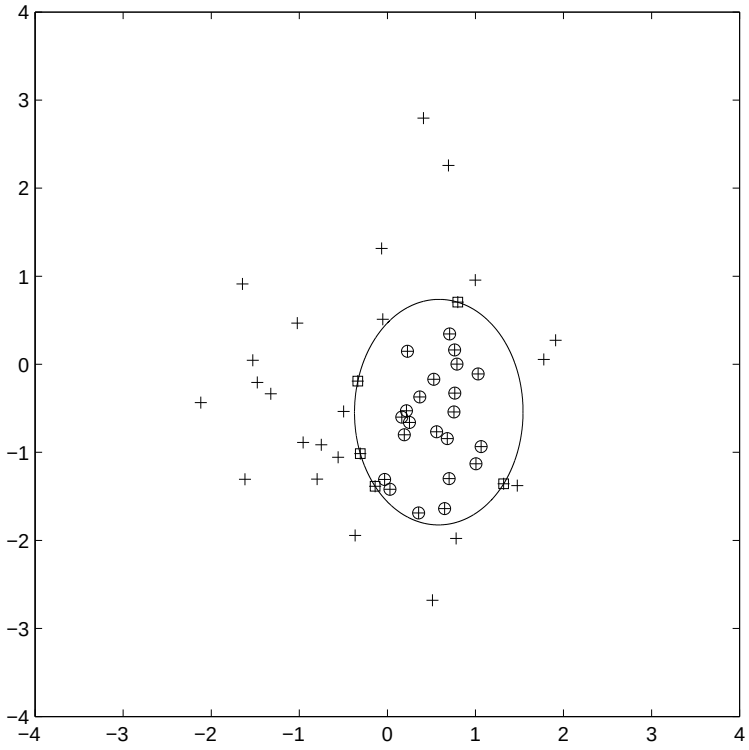


Figure 4.12: A quadratic defined by five points that selects MCD_{25}.

the quadratic still passes through the point with maximum Mahalanobis squared distance, marked with a square in Figure 4.11. This is a linear constraint, so we are not limited in how we adjust the remaining four degrees of freedom.* As soon as the quadratic touches another point, an additional constraint is added such that the quadratic must now also pass through this point. This continues until all five degrees of freedom have been set by five points from the sample. One possible outcome of this procedure for the MCD of data in Figure 4.11 is shown in Figure 4.12, where each of the five points that define the quadratic† is marked with a square. Notice that three of these are part of the MCD subset.

As mentioned above, the original quadratic is negative if evaluated for a point internal to the ellipse. The continuous variation of the coefficients means that in order for the quadratic to become positive, it must pass through zero, but if this happens, then the quadratic has touched the point in question, and it becomes one

*The only assumption is that the points are in general quadric position. This is equivalent to assuming a non-singular matrix in (4.11).

†In this case it is another ellipse, but if $B'^2 - 4A'C'$ is positive then a hyperbola is formed.

of the five points that will define the final quadratic. As a result, this new quadratic still selects the MCD subset, as can be seen in Figure 4.12. The status of the five points that fall on the boundary is uncertain. All we know is that at least one of them (the point the original ellipse passed through) is part of the subset.*

4.5.2 Description

If we are given the indices $\{i_1, \ldots, i_5\}$ of the five points that define the quadratic of Figure 4.12, then we can find its coefficients by solving the linear system

$$\begin{bmatrix} x_{i_1}y_{i_1} & y_{i_1}^2 & x_{i_1} & y_{i_1}1 \\ x_{i_2}y_{i_2} & y_{i_2}^2 & x_{i_2} & y_{i_2}1 \\ x_{i_3}y_{i_3} & y_{i_3}^2 & x_{i_3} & y_{i_3}1 \\ x_{i_4}y_{i_4} & y_{i_4}^2 & x_{i_4} & y_{i_4}1 \\ x_{i_5}y_{i_5} & y_{i_5}^2 & x_{i_5} & y_{i_5}1 \end{bmatrix} \begin{bmatrix} B' \\ C' \\ D' \\ E' \\ F' \end{bmatrix} = - \begin{bmatrix} x_{i_1}^2 \\ x_{i_2}^2 \\ x_{i_3}^2 \\ x_{i_4}^2 \\ x_{i_5}^2 \end{bmatrix}. \tag{4.11}$$

MCD-BF goes through all $\binom{N}{5}$ subsets and solves (4.11) for each. The solution is used to evaluate the quadratic for the remaining $N-5$ points to determine the number τ, which are selected. For the quadratic to have been derived from an MCD subset of size n,

$$n - 5 \leq \tau < n, \tag{4.12}$$

which simply says that at least one, and at most all five, of the defining points may be part of the $\binom{N}{n}$-subset. If this holds, then each of the $\binom{5}{n-\tau}$ subsets of the five defining points are combined with the τ points to form $\binom{N}{n}$ subsets. The covariance determinant of each of these is then determined, with the lowest being recorded. Otherwise the next $\binom{N}{5}$ subset is constructed and a new quadratic is considered.

The processing of the $\binom{N}{5}$ subsets is $O(N^5)$, and for each of these, finding τ involves evaluating the quadratic for at most $N-5$ points. This is linear, so the algorithm is $O(N^6)$. Although we could not decrease the order of this complexity, a very significant constant improvement is realized if one ignores those quadratics with $C' < 0$. We know that $C > 0$, so we include the constraint that if it becomes zero at any stage during the construction of the quadratic, then it is fixed there. If this happens, then only four points are needed to define the quadratic, and a 4×4 system similar to (4.11) is solved for the remaining coefficients. An otherwise identical procedure is followed as for the $\binom{N}{5}$ subset search. In fact, an examination of the pseudocode of our implementation in Figure 4.13 reveals that both are carried

*The original authors didn't notice this, and regard their status as completely unknown.

out by generalized $\binom{N}{f}$ subset search code. The complexity of the $f = 4$ code is one degree less than for $f = 5$, so the additional work required quickly becomes negligible. Simulations show that in an uncontaminated sample, an $\binom{N}{5}$ subset quadratic has a negative C' about half the time.

4.5.3 Comments

The subset tree principle that produced the $\binom{N}{n}$ subsets of MCD-BB and MCD-SW is again used to search through the various subsets in MCD-BF. We have omitted the routine from the pseudocode since it is covered in detail in section 4.2. The variable i is used to store the indices of the $\binom{N}{f}$ subset tree, and r for the $\binom{f}{n-\tau}$ subset tree.

The `find_selected` routine does most of the work of MCD-BF. The subset tree method of subset construction results in the addition of a new row to (4.11) at each level. This is reduced to Gauss-Jordan form and stored. The next row is treated similarly, and uses the Gauss-Jordan form from the previous level. This is a very efficient way of repeatedly solving the system, since it avoids having to reduce the matrix from scratch for each subset. The routine ends with the evaluation of the quadratic for the $N - f$ points. If it becomes known that (4.12) will not be satisfied, then it returns immediately.

Only a small fraction of the solutions to (4.11) yield the MCD subset. For data X_{50}, just 88 of the $\binom{50}{f}$ subsets produce the MCD, with all of these being for $f = 5$. The fact that many different $\binom{N}{f}$ subsets can produce the same $\binom{N}{n}$ subset is a redundancy inherent to MCD-BF. Finding the number of unique $\binom{N}{n}$ subsets entails code that compares each new subset to all the others already found. This is efficiently implemented with a binary search tree, but even so, it slows the algorithm down enough for it to be excluded by default. Define the macro `UNIQUE` and recompile to include this code. Data X_{50} contains 43843 unique subsets of size n that are quadratic selectable. Some of these, like the example shown in Figure 4.14, are not even convex complete and could therefore not even have been obtained from an ellipse selectable subset. This suggests that there may be further ways to improve MCD-BF.

```
for f = 4 to 5
   i := {1, 2, ..., f}
   do
      Call find_selected
      if τ ≠ n
         r := {1, ..., n − τ}
         do
            Calculate |S|
            if |S| is smallest found
               Store it and the subset
            endif
         until next_subset(f, n − τ) returns 0
      endif
   until next_subset(N, f) returns 0
next f

find_selected:
   while l <= f
      Add row [x_{i_l} y_{i_l}  y²_{i_l}  x_{i_l}  y_{i_l}  1  | − x²_{i_l}]
      and reduce the matrix to Gauss-Jordan.
      if f = 5 and C' < 0
         return n
      endif
      l := l + 1
   endwhile
   l := 1
   for j = 1 to N
      if j = i_l
         if n − f − τ > N − j
            return n
         endif
         l := l + 1
      elseif A'x_j² + B'x_j y_j + C'y_j² + D'x_j + E'y_j + F' < 0
         τ := τ + 1
         if τ = n
            return n
         endif
      elseif n − f − τ > N − j
         return n
      endif
   next j
return τ

next_subset:
   Update the subset combination array (i or r), and
   return 0 once all subsets have been considered.
end
```

Figure 4.13: Pseudocode for MCD-BF (Source code in Appendix B.5)

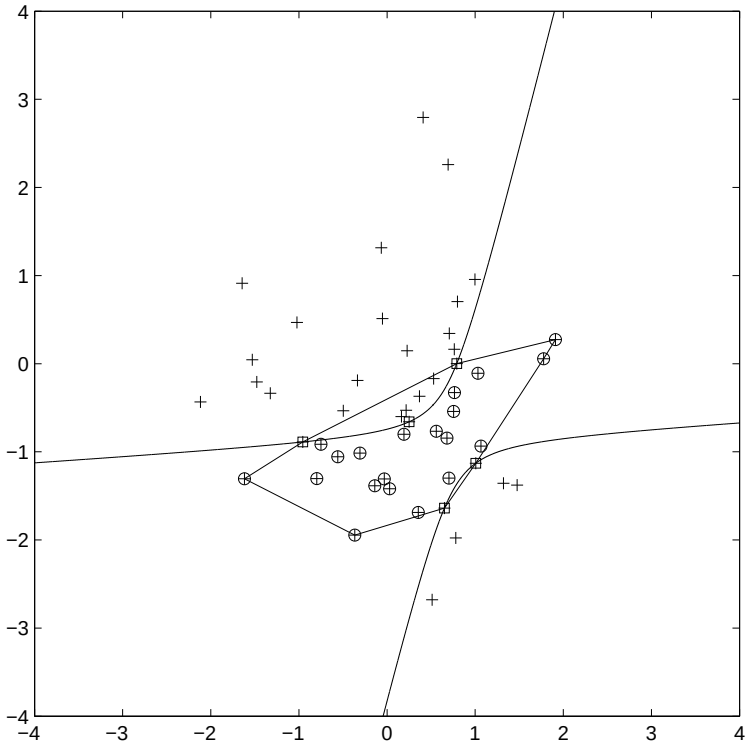


Figure 4.14: Quadratic selectable subset that is not convex complete

4.6 Testing of the exact MCD algorithms

4.6.1 Introduction

The four algorithms covered in the preceding sections were programmed in C as MATLAB modules, and tested on a 1.7 GHz Celeron machine. To simplify the tests, we used even N and found only $\text{MCD}_{N/2}$, allowing for a study of time complexity as function of N. The samples were distributed as $N(\mathbf{0}, \mathbf{I})$, but because the MCD is affine equivariant, this is not a limitation. The inclusion of contaminants is never going to be satisfactory because of the infinite number of possible configurations of outliers and clusters, so none were included. All but MCD-BF displayed erratic running times, so in order to produce smoother performance curves we took the average of 10 samples for each size.

4.6.2 Run-time performance

The results are shown in Figure 4.15 with the logarithm of the CPU time against sample size. The first noteworthy feature is that MCD-BB is clearly exponential, which limits it to samples of size 50. MCD-SW is more promising, and exhibits behaviour that might be polynomial. The same can be said of the slightly faster MCD-EXT, but for samples larger than about 30, MCD-BF is in a league of its own. Its smooth curve may be attributed to the predictable and systematic way it searches through the sample that depends little on the configuration of the observations.

The possibly polynomial characteristics of MCD-SW and MCD-EXT deserve further investigation. If they are indeed polynomial, then this will be revealed in a log-log plot as a linear relationship. In order to confirm that MCD-BF *is* polynomial, we include its running times in the same plot, shown in Figure 4.16. It is unfortunately not possible to claim with any confidence that either MCD-SW or MCD-EXT is polynomial, since both curves are clearly concave and although they may be asymptotically polynomial, it is not obvious when this might occur. We nevertheless estimate their respective degrees as 10 and 8 in the vicinity of the last few observations. This was done by a simple linear regression of these final points, and an extrapolation of these lines is also plotted. As expected, MCD-BF is a well-behaved polynomial with the degree estimated at $5\frac{2}{3}$. This is slightly less than $O(N^6)$, and is probably due to the ostensibly linear search through the $N - f$ points. (See section 4.5.2) This is aborted early if at all possible, and depends on the number of points τ that are selected. It is quite possible that this makes the

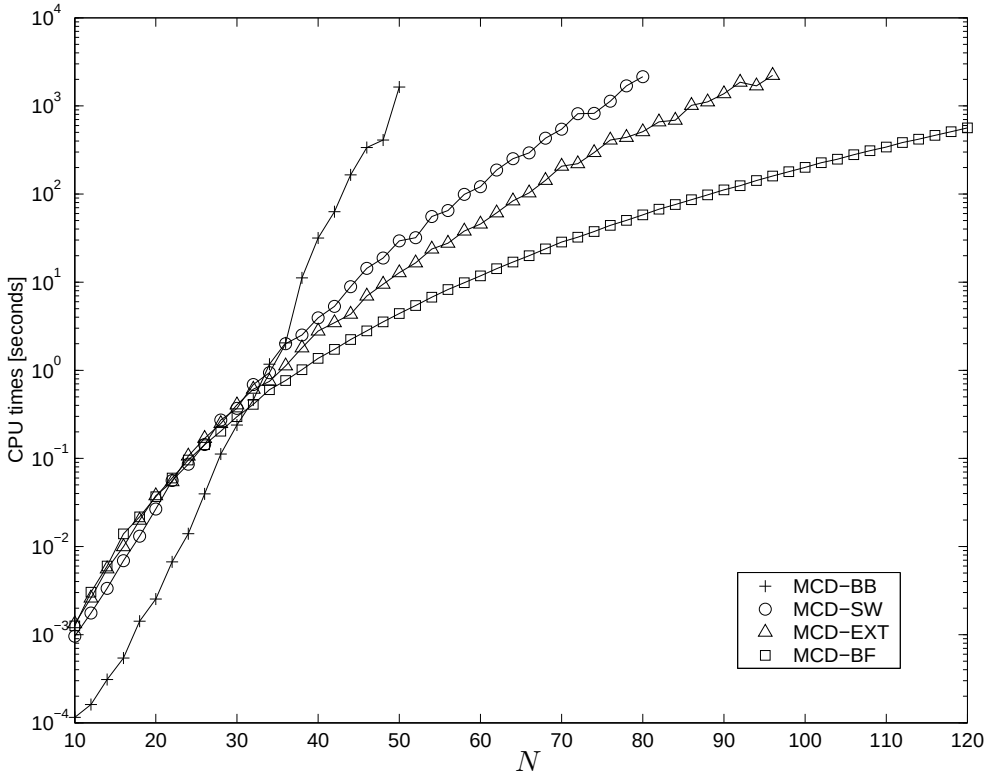


Figure 4.15: Running times of the four exact algorithms

search sub-linear. It may also be that N is simply not large enough for the results to take on the limiting behaviour.

4.6.3 Comments

Pesch [20] constructs a number of hybrid algorithms (for example combining MCD-BB and MCD-SW) but we feel that his results show that none of the combinations he tested compliment each other sufficiently well to justify their union.

The outward testing procedure requires a *set* of inlier subsets from size h onward. MCD-BB and MCD-SW are based on a subset tree constructed for a particular subset size, so calculating a set of MCD subsets requires running the algorithm from scratch for each of them. MCD-EXT and MCD-BF are not bound to a subset size from the outset, and are ideally suited to modification that would produce all the MCD subsets needed. This would require minimal extra computation, and for MCD-BF it would not change the complexity of the algorithm.

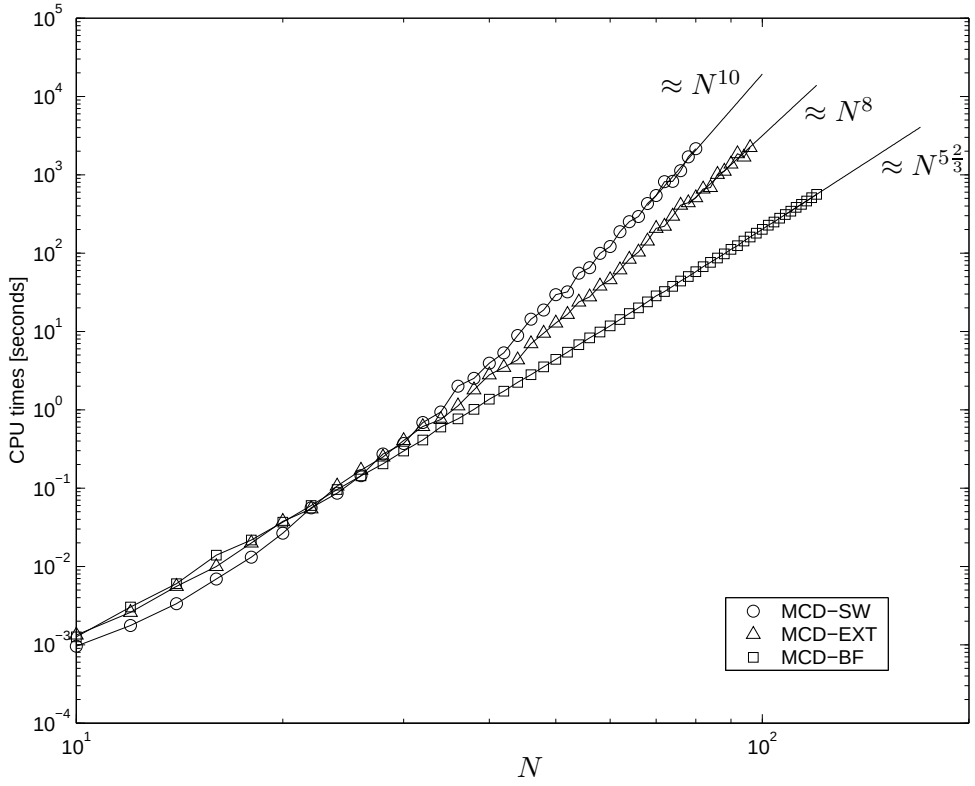


Figure 4.16: Log-log plot for examining the polynomial-like run times

4.7 The heuristic Fast-MCD algorithm

4.7.1 Theory

Fast-MCD, as published by Rousseeuw and van Driessen [24], relies on a procedure whose origin Pesch [20] traces back to Gnanadesikan and Kettenring [11]. Given an estimate of the mean and covariance matrix produced by a certain subset of size n, it is sometimes possible to find another with smaller covariance determinant. This is done by simply calculating the Mahalanobis distances of the entire sample, and defining a new subset as those n points which produced the smallest of these. If the resulting subset is the same as before, then nothing is achieved, otherwise the new subset will have a covariance determinant smaller than the initial one. The basis of this is the theorem which states that if

$$\sum_{\mathbf{x}_i \in I_n} (\mathbf{x}_i - \bar{\mathbf{x}})^T \mathbf{S}^{-1} (\mathbf{x}_i - \bar{\mathbf{x}}) > \sum_{\mathbf{x}_i \in \tilde{I}_n} (\mathbf{x}_i - \bar{\mathbf{x}})^T \mathbf{S}^{-1} (\mathbf{x}_i - \bar{\mathbf{x}}), \tag{4.13}$$

then $|\tilde{\mathbf{S}}| < |\mathbf{S}|$, where $\tilde{\mathbf{S}}$ and $\mathbf{S}$ are the covariance matrices produced by $\tilde{I}_n$ and I_n, respectively. The mean of I_n is denoted by $\bar{\mathbf{x}}$. Because we chose the points corresponding to the n smallest Mahalanobis distances, we can be sure (assuming the subset has changed) that their sum is strictly less than that of the original subset, which guarantees the decrease in the covariance determinant.

We now prove this for the special case where $\mathbf{S} = \mathbf{I}$ and $\bar{\mathbf{x}} = \mathbf{0}$, and note that extension to the general case may be achieved by using the transformation $\mathbf{y}_i = \mathbf{S}^{-\frac{1}{2}}(\mathbf{x}_i - \bar{\mathbf{x}})$. For details of this transform see Fukunaga [9] (p 28 ff), where it is referred to as a *whitening* transform. The outline of this proof is from Pesch [19] (pp 8-9). The special case simplifies (4.13) to

$$\sum_{\mathbf{x}_i \in I_n} \mathbf{x}_i^T \mathbf{x}_i > \sum_{\mathbf{x}_i \in \tilde{I}_n} \mathbf{x}_i^T \mathbf{x}_i. \tag{4.14}$$

The mean of $\tilde{I}_n$ is denoted by $\bar{\tilde{\mathbf{x}}}$, and will in general not be zero. If the covariance matrix of $\tilde{I}_n$ were calculated using a mean of $\mathbf{0}$ and denoted by $\tilde{\mathbf{S}}_\mathbf{0}$, then the following relationship between it and the true covariance matrix holds,

$$\tilde{\mathbf{S}}_\mathbf{0} = \frac{1}{n} \sum_{\mathbf{x}_i \in \tilde{I}_n} \mathbf{x}_i \mathbf{x}_i^T$$

$$
\begin{aligned}
&= \frac{1}{n} \sum_{\mathbf{x}_i \in \tilde{I}_n} \left((\mathbf{x}_i - \bar{\bar{\mathbf{x}}}) + \bar{\bar{\mathbf{x}}} \right) \left((\mathbf{x}_i - \bar{\bar{\mathbf{x}}}) + \bar{\bar{\mathbf{x}}} \right)^T \\
&= \frac{1}{n} \sum_{\mathbf{x}_i \in \tilde{I}_n} (\mathbf{x}_i - \bar{\bar{\mathbf{x}}})(\mathbf{x}_i - \bar{\bar{\mathbf{x}}})^T + \bar{\bar{\mathbf{x}}}\bar{\bar{\mathbf{x}}}^T \\
&= \tilde{\mathbf{S}} + \bar{\bar{\mathbf{x}}}\bar{\bar{\mathbf{x}}}^T. \qquad (4.15)
\end{aligned}
$$

We used the fact that $\sum_{\mathbf{x}_i \in \tilde{I}_n} (\mathbf{x}_i - \bar{\bar{\mathbf{x}}}) = \mathbf{0}$ to simplify the above expression. Taking the determinant of both sides produces

$$
|\tilde{\mathbf{S}}_\mathbf{0}| = |\tilde{\mathbf{S}}| \left[1 + \bar{\bar{\mathbf{x}}}^T \tilde{\mathbf{S}}^{-1} \bar{\bar{\mathbf{x}}} \right] \geq |\tilde{\mathbf{S}}|, \qquad (4.16)
$$

where the inequality follows from the fact that $\tilde{\mathbf{S}}^{-1}$ is positive definite.

The matrix $\tilde{\mathbf{S}}_\mathbf{0}$ is also positive definite, so its determinant is the product of its positive eigenvalues. It is also well known that the trace of a matrix is equal to the sum of its eigenvalues, so the inequality

$$
|\tilde{\mathbf{S}}_\mathbf{0}| \leq \left(\frac{\text{tr}\left(\tilde{\mathbf{S}}_\mathbf{0}\right)}{p} \right)^p, \qquad (4.17)
$$

follows if one recalls the inequality relating the geometric and arithmetic means of a set of positive real numbers ([12], p 16).

We now establish an upper bound on $\text{tr}\left(\tilde{\mathbf{S}}_\mathbf{0}\right)$.

$$
\begin{aligned}
\text{tr}\left(\tilde{\mathbf{S}}_\mathbf{0}\right) &= \text{tr}\left(\frac{1}{n} \sum_{\mathbf{x}_i \in \tilde{I}_n} \mathbf{x}_i \mathbf{x}_i^T \right) \\
&= \text{tr}\left(\mathbf{S} - \frac{1}{n} \left(\sum_{\mathbf{x}_i \in I_n} \mathbf{x}_i \mathbf{x}_i^T - \sum_{\mathbf{x}_i \in \tilde{I}_n} \mathbf{x}_i \mathbf{x}_i^T \right) \right) \\
&= \text{tr}\left(\mathbf{I}\right) - \frac{1}{n} \left(\sum_{\mathbf{x}_i \in I_n} \text{tr}\left(\mathbf{x}_i^T \mathbf{x}_i\right) - \sum_{\mathbf{x}_i \in \tilde{I}_n} \text{tr}\left(\mathbf{x}_i^T \mathbf{x}_i\right) \right) \\
&= p - \frac{1}{n} \left(\sum_{\mathbf{x}_i \in I_n} \mathbf{x}_i^T \mathbf{x}_i - \sum_{\mathbf{x}_i \in \tilde{I}_n} \mathbf{x}_i^T \mathbf{x}_i \right) \qquad (4.18)
\end{aligned}
$$

It follows from (4.14) that $\text{tr}\left(\tilde{\mathbf{S}}_\mathbf{0}\right) < p$. Combining this inequality with those of

(4.16) and (4.17) yields

$$|\tilde{\mathbf{S}}| \leq |\tilde{\mathbf{S}}_\mathbf{0}| \leq \left(\frac{\operatorname{tr}\left(\tilde{\mathbf{S}}_\mathbf{0}\right)}{p} \right)^p < \left(\frac{p}{p} \right)^p = 1 = |\mathbf{S}|, \tag{4.19}$$

which completes the proof.

A useful interpretation of this result is that selecting the subset with the smallest sum of the Mahalanobis distances favours those subsets that are tightly grouped. Intuitively, the MCD subset is therefore likely to be a dense collection of points in space, and prompted Rousseeuw and van Driessen [24] to call this a *C-step*, where C stands for "concentration". Furthermore, a corollary of this result is the proposition that a bivariate MCD subset is selectable by an ellipse (see section 4.5), since if the MCD subset is not selectable by its own covariance ellipse, then it means that any foreign point within it has a smaller Mahalanobis distance than the point defining the ellipse. Exchanging this point with any foreign one will reduce the sum of the Mahalanobis distances, and the new subset will have a reduced covariance determinant. This is a contradiction, so we conclude that the assertion is correct.

4.7.2 Description

The basic form of an algorithm from the above result is self-evident. Generate a random subset I_n from the sample and find $\tilde{I}_n$. Use this subset as the next I_n and repeat the C-step until $I_n = \tilde{I}_n$. Pesch [20] calls this search for a local minimum *iterative trimming*, and it can be repeated some specified number of times (known as the number of restarts), with the minimum of these chosen as the MCD estimate. The number of restarts necessary is obviously a crucial factor, and it is addressed in section 4.8.2. Finding the n smallest Mahalanobis distances for constructing $\tilde{I}_n$ is a selection problem, and does not require that the distances are ordered. It may be solved in $O(N)$ time, so this and the calculation of all the Mahalanobis distances makes each C-step a linear operation.

The basic algorithm as just outlined works well for a sample containing one main cluster and possibly a few scattered outliers. If it is applied to one composed of two similar clusters of the same size, then the random subset will almost certainly contain a fair number of points from both, and this is unlikely to change during the minimization steps. The covariance estimate of the random subset has to be more or less representative for the minimization steps to find the MCD. We need to maximize the probability that the subset contains only points from one of the

clusters, and the easiest way to achieve this is to select fewer points. Fast-MCD randomly selects just $p+1$ points, which is the minimum required for a non-singular covariance matrix. If these are all from only one cluster, then it is likely that they are evenly scattered within it, and this produces an estimate of the covariance matrix which is good enough for the minimization steps to stand a chance of finding the MCD. This is a vital part of what makes the algorithm practical. Rousseeuw and van Driessen [24] remark that the probability of finding at least one such "clean" subset amongst m random subsets is

$$\Pr\{\text{At least one clean } p+1 \text{ subset}\} = 1 - (1 - (1-\kappa)^{p+1})^m. \tag{4.20}$$

We consider this probability again later.

4.7.3 Comments

Fast-MCD as described in [24] contains two other relatively mundane modifications to the procedure described. The first of these is the termination of the iterative trimming after just two iterations, with some proportion of the best of these being retained and allowed to converge. The argument is that if convergence is not happening fast enough it is likely that this is an indication of a "dirty" starting subset that is not likely to converge to the MCD. The example they use to illustrate this is made up of two clusters, with the one slightly larger than the other being treated as the "good" data. We are wary of techniques like these that rely on the type of contamination to speed up the algorithm. It may work in practice, but in the next section we continue in our puritan approach to testing algorithms by using only uncontaminated data. The reader is reminded of our comment on this topic in section 4.6.1.

The second modification deals with large data sets and how partitioning them into a hierarchy of mutually exclusive, random subsets may be used to make the problem more manageable. We feel that this is a somewhat generic technique for dealing with large data sets, and again stress that while it is undoubtedly effective in practice, the introduction of more arbitrarily chosen parameters (the number and size of the subsets and levels of subdivision) further complicates an analysis of the algorithm. Whether or not these are generally suitable is also unknown, and the authors concede in [24] that their "choices were based on various empirical trials", and that the user may wish to change the defaults. This uncertainty is further justification for testing a basic version of the algorithm rather than a specific

variation of it.

4.8 Empirical analysis of Fast-MCD

4.8.1 Introduction

Rousseeuw and van Driessen [24] make the implicit assumption that with a suitably high number of restarts, Fast-MCD will find a good approximation to the MCD. They do not concern themselves with whether this is likely to be the exact MCD, and it was Pesch [20] who first investigated this using his exact algorithms. His measure of the efficacy of the heuristic algorithms was based on the percentage difference in the covariance determinant between the final output from the heuristic algorithm and the MCD, in addition to a consideration the probability of finding the exact MCD. We focus on the latter, and in particular on how it might vary with N.

Our emphasis is on analysing a single iterative trimming convergence rather than the final output produced by some arbitrary number of restarts. We investigate the distribution of the number of iterations needed to reach a local minimum, and also the probability that a random starting subset will find the MCD. We term this the *global convergence probability* and denote it with ρ. Knowledge of this allows us to make predictions as to how many restarts are actually needed in the first place, given that one requires the exact MCD with some predetermined degree of confidence. We were also able to test larger sample sizes than Pesch by using MCD-BF. Specifically, Pesch was limited to $N = 100$ for half-sample MCD subsets, and our implementation of MCD-BF takes just a few hours for $N = 200$, on the same machine used for testing the exact algorithms in section 4.6.2. For empirical analysis, a single MCD subset of a large sample is not very useful, so we prefer to limit ourselves to $N = 160$ so that many samples of this size can be examined and averages taken to establish trends.

4.8.2 Testing

Our simulations used samples from $N(\mathbf{0}, \mathbf{I})$ in sizes ranging from 10 to 160 at intervals of 10. Twenty such samples of each size were generated and their MCD subsets found using MCD-BF. All results presented are from the average over these twenty. An exhaustive test of all $\binom{N}{3}$ possible starting subsets for an iterative trim was then conducted, with the number of iterations and of course whether or not it found the MCD, being recorded. The distribution of the number of iterations is shown in

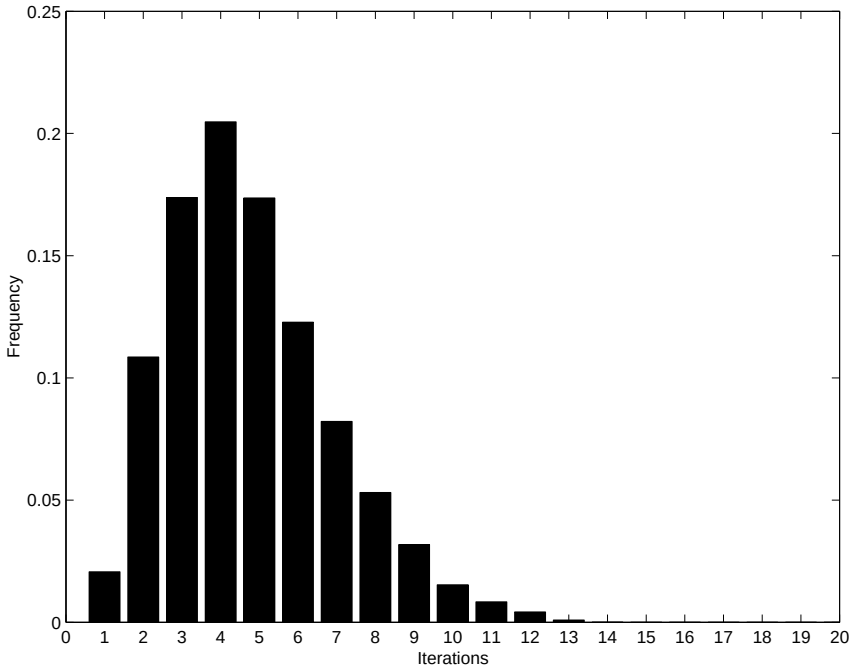


Figure 4.17: The number of iterations required for $N = 70$

Figure 4.17 for $N = 70$. Although not shown here, we did confirm that that the only difference between those iterations that ultimately found the MCD and those that didn't, is that the former tended to take a little longer, which shifts the distribution slightly to the right. For instance, all of the few subsets (so few, in fact, that the bar is not visible in Figure 4.17) that needed 14 iterations found the MCD.

The average number of iterations until convergence as a function of N is shown in Figure 4.18. Substantial growth is observed, and although this appears to be sub-linear (perhaps logarithmic), it is certainly significant. We consider the absence of any mention of it a surprising omission in the literature, especially since it doesn't even require calculating the MCD. Thus, it cannot be claimed that Fast-MCD is linear for a constant number of restarts, and the importance that Pesch [20] (p 114) and Rousseeuw and van Driessen [24] (p 215 and p 220) place on each C-step being linear instead of $O(N \log N)$ could therefore be misleading.

We now turn our attention to the estimates $\hat{\rho}$ of the global convergence probability shown in Table 4.1. Note in particular the lack of any discernible dependence of ρ on N, which is a very desirable property indeed. It means that we can have the same degree of confidence in the result after a given number of restarts, irrespec-

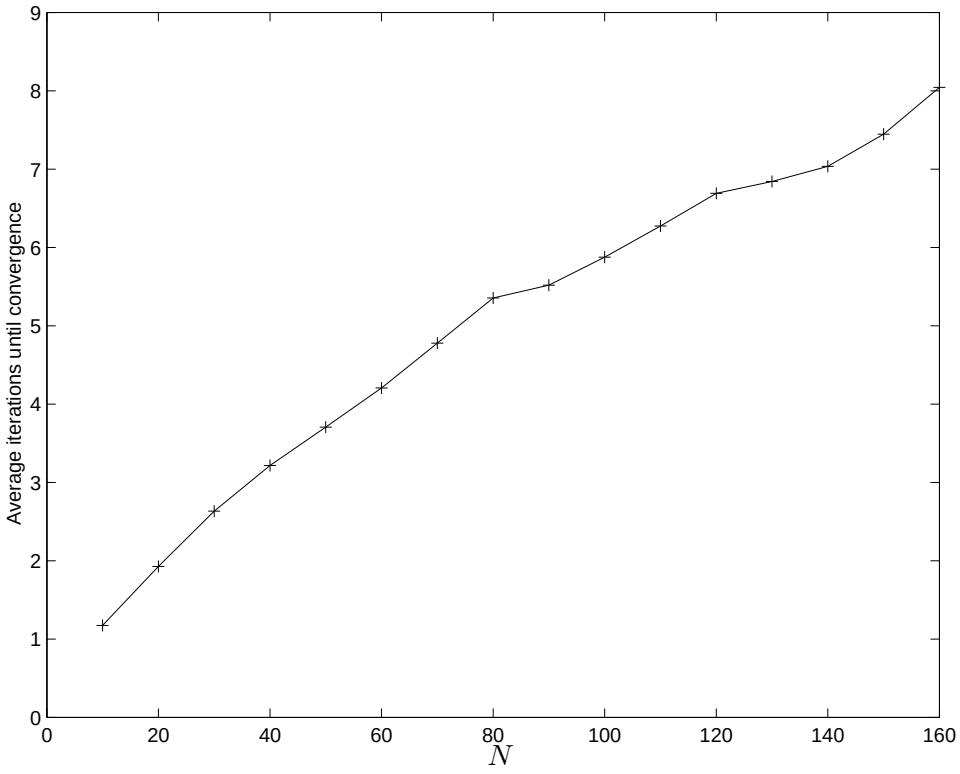


Figure 4.18: The average number of iterations as a function of N

N	$\hat{\rho}$	N	$\hat{\rho}$	N	$\hat{\rho}$	N	$\hat{\rho}$
10	0.122	50	0.116	90	0.159	130	0.167
20	0.139	60	0.122	100	0.088	140	0.070
30	0.149	70	0.114	110	0.135	150	0.094
40	0.111	80	0.182	120	0.091	160	0.152

Table 4.1: Estimates of ρ for bivariate data

tive of the sample size and the fact that the number of iterations for each restart increases.

4.8.3 Comments

We can now extend (4.20) to represent the probability of actually finding the MCD of a bivariate sample after m restarts,

$$\Pr\{\text{Bivariate MCD is found}\} = 1 - \left(1 - \rho(1-\kappa)^3\right)^m, \tag{4.21}$$

and use this to calculate the number of restarts necessary to be $(1-\alpha)\times 100\%$ sure that the result is the MCD with

$$m \geq \frac{\log(\alpha)}{\log\left(1-\rho(1-\kappa)^3\right)}. \tag{4.22}$$

Notice that we have made two conservative assumptions in arriving at this expression. One is that it is impossible for a starting subset that isn't clean to converge to the MCD, and the other is that the worst possible proportion of contamination is realized. Even if the latter does occur, but the contamination takes the form of outliers evenly scattered around the good data, then there is no reason that starting subsets containing one or more of these outliers shouldn't have as good a chance of converging to the MCD as a clean one. This is mitigated to some extent in that a clean starting subset could conceivably be affected by contaminants, and is therefore slightly less likely to find the MCD than if no contaminants were present.

As an example we take $\kappa = \frac{1}{2}$ and $\rho = 0.126$ (the average of the values in Table 4.1) and find that 189 restarts are needed for a 95% confidence of obtaining the MCD. It is worth mentioning that the risk of not finding the MCD is not a serious one. If more than n inliers are present, then many n-subsets of these will be perfectly adequate as an inlier subset. It is more likely that one of these will be discovered if the MCD is not found, rather than a really bad estimate containing

outliers. What *is* important is the sensitivity of (4.22) to changes in the proportion of contamination. If $\kappa = 0.1$, then just 32 restarts are required.

4.9 Alternatives to the MCD

The somewhat detailed discourse on the MCD should not distract us from the objective of this chapter, which is to find a *set* of inlier subsets, not just one. The repeated use of Fast-MCD to estimate the MCD for all sizes from h onwards, though far faster than any of the exact algorithms, will still take some time on a large data set. This section evaluates the potential of the EDR and Recovery procedures applied so successfully to univariate data in chapter 2.

4.9.1 Extreme Deviate Removal (EDR)

In exactly the same matter as for univariate data, we consider the removal of the observation that maximizes the likelihood of the remaining sample. $\mathbf{S}_d$ is the MLE of the covariance matrix of $\{I_n \backslash \mathbf{x}_d\}$, and some matrix algebra shows that

$$\mathbf{S}_d = \frac{n}{n-1}\left(\mathbf{S} - \frac{1}{n-1}(\mathbf{x}_d - \bar{\mathbf{x}})(\mathbf{x}_d - \bar{\mathbf{x}})^T\right). \tag{4.23}$$

Taking the determinant of the above yields

$$\begin{aligned} |\mathbf{S}_d| &= \left(\tfrac{n}{n-1}\right)^p |\mathbf{S} - \tfrac{1}{n-1}(\mathbf{x}_d - \bar{\mathbf{x}})(\mathbf{x}_d - \bar{\mathbf{x}})^T| \\ &= \left(\tfrac{n}{n-1}\right)^p |\mathbf{S}| \left(1 - \tfrac{1}{n-1}(\mathbf{x}_d - \bar{\mathbf{x}})^T \mathbf{S}^{-1}(\mathbf{x}_d - \bar{\mathbf{x}})\right). \end{aligned} \tag{4.24}$$

It is seen that removal of the observation with the greatest Mahalanobis distance results in the subset with the smallest covariance determinant.

This method is used by Caroni and Prescott [6] and also justified with likelihoods arguments, though in the form of the ratio $|\mathbf{S}_d|/|\mathbf{S}|$, which from (4.24) is clearly equivalent. The problems of masking and swamping are mentioned in the introduction of [6], but only in relation to how these may negatively affect the test statistic, not the inlier subsets. In fact, they do not formalize the notion of inlier subsets at all.

In the univariate case it was seen that only in special cases was EDR unreliable. In the multivariate case EDR is vulnerable to swamping if a cluster of outliers is present (see Figures 1.1 and 1.2) so we strongly discourage its use.

4.9.2 MCD with Recovery

In the univariate case the concept of recovery was used to rectify the problems that might have arisen because of swamping during EDR. Applying EDR repeatedly on multivariate data until only $p+1$ points remain is futile (even if these are all inliers), since the estimate of the covariance matrix with so few points is almost guaranteed to be useless, making useful recovery of points from such a seed subset impossible.

The MCD, on the other hand, is an ideal seed subset because it serves as a good robust estimate of the true covariance matrix, which greatly diminishes the problems of masking and swamping. Using the same maximum likelihood principle when adding a new observation as for univariate data, and using $\mathbf{S}_a$ to denote the MLE of the covariance matrix of $\{I_n, \mathbf{x}_a\}$, we find that the recovery version of (4.23) is

$$|\mathbf{S}_a| = \left(\tfrac{n}{n+1}\right)^p |\mathbf{S}| \left(1 + \tfrac{1}{n+1}(\mathbf{x}_a - \bar{\mathbf{x}})^T \mathbf{S}^{-1} (\mathbf{x}_a - \bar{\mathbf{x}})\right). \tag{4.25}$$

This shows that the observation with smallest Mahalanobis distance must be added to maximize the likelihood of the new sample. That this nearest neighbouring point be included is an intuitively pleasing result, and it is surprising that such an obvious reversal of EDR has not, to our knowledge, been explored before. It is worth noting that this observation does not always correspond to that with maximum Mahalanobis distance in I_{n+1}, so calculating the test statistic involves first finding this point.

Even if the MCD doesn't provide an especially good estimate, it is perfectly adequate as long as recovery proceeds though all the inliers before starting on the outliers. The only situation in which the estimate could conceivably be bad enough for this not to occur, is if the subset size is too small. Even then, the estimate would have to be egregious for the nearest point to be an outlier. This is illustrated by returning to the example of MCD_{12} of X_{50}, redisplayed here in Figure 4.19. An outlier* has been placed in a deliberately unfavourable position directly in line with the main axis of the ellipse, so as to make it appear as close as possible to MCD_{12} while still being distant enough under the correct metric for it not to lose its status as an outlier. Despite our best efforts to lead the recovery astray, there are still two inliers not already part of MCD_{12} that are nearer to it than the outlier.

As more points are added the estimate improves, and can literally be said to *recover* from the poor start. To establish just how effective this is, we compared each recovered subset to the MCD of the same size. The difference between the two

*None of the MCD subsets were affected by it

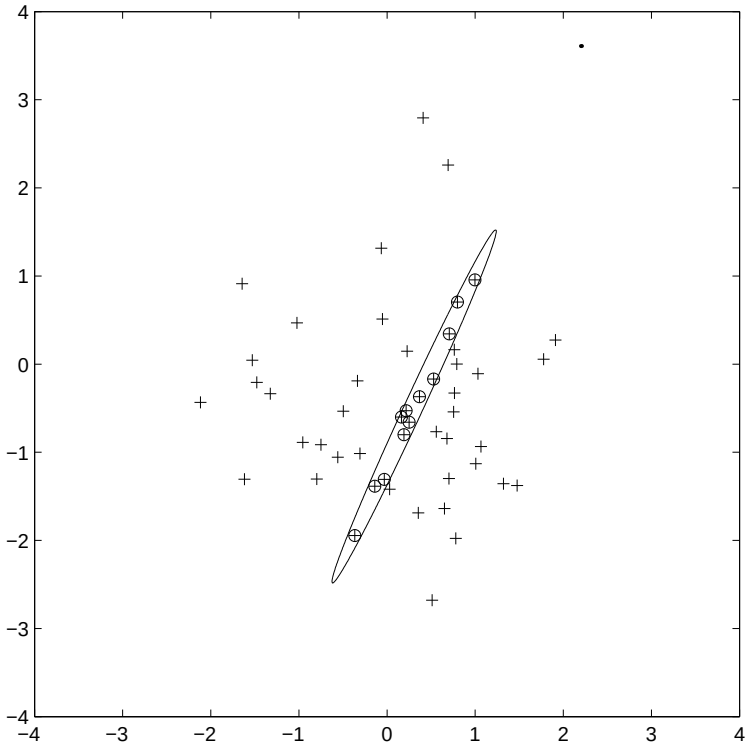


Figure 4.19: Recovery from MCD_{12} with an unpleasant outlier ($\cdot$) present

n	δ	n	δ	n	δ	n	δ	n	δ	n	δ
12	0	19	2	26	1	33	1	40	0	47	0
13	0	20	0	27	3	34	2	41	0	48	0
14	0	21	1	28	3	35	2	42	2	49	0
15	0	22	0	29	3	36	1	43	0	50	0
16	0	23	0	30	2	37	1	44	0		
17	3	24	3	31	3	38	0	45	0		
18	3	25	2	32	2	39	0	46	0		

Table 4.2: The discrepancy δ between the MCD and recovered subsets of X_{50}

is measured by $\delta = n - |\text{MCD}_n \cap I_n|$, and is shown in Table 4.2. The results are impressive, with many of the recovered subsets being identical to the MCD. For the rest, the discrepancy is never more than 3, and subsets I_{43} onward are all identical to the MCD. It is during this stage that a transition from inliers to outliers may occur, so the equivalence of the two is crucial if the same conclusions are to be drawn.

The updating of the inverse of the covariance matrix after the addition of $\mathbf{x}_a$ may be done using the recovery analogue of the formula in [6] (p 358),

$$\mathbf{S}_a^{-1} = \left(\tfrac{n+1}{n}\right)\left(\mathbf{S}^{-1} - \frac{\mathbf{S}^{-1}(\mathbf{x}_a - \bar{\mathbf{x}})(\mathbf{x}_a - \bar{\mathbf{x}})^T\mathbf{S}^{-1}}{n+1+z_a(n-1)}\right), \tag{4.26}$$

where z_a is the Mahalanobis distance of $\mathbf{x}_a$ as defined by (5.1). This makes for very rapid recovery from the seed subset.

4.10 Example - Artificial Normal data

A more thorough test of recovery is needed to investigate its behaviour with larger data sets of higher dimension. An artificial data set with $p = 5$, $N = 100$, and distributed as $N(\mathbf{0}, \mathbf{I})$, was generated. The MCD subsets from size $n = 25$ onward were estimated using Fast-MCD with $m = 10000$. We don't know what the global convergence probability for dimension five is, but we presume that such a large number of restarts is enough to find the exact MCD with a reasonably high probability, particularly since we know that $\kappa = 0$.

The discrepancy fraction $\frac{\delta}{n}$ is plotted in Figure 4.20 as a function of n for the I_n recovered from MCD_{25} and MCD_{50}. Of course, they both start out at zero, but those recovered from MCD_{25} suddenly jump to nearly $\frac{2}{3}$ after just a few steps. It improves gradually soon afterward, but remains high until about $n = 80$. This behaviour is simply the result of the MCD having not yet "settled down" for low n. This is still the case in the vicinity of $n = 55$, where even the subsets recovered from MCD_{50} differ momentarily to quite a large degree. Apart from this, the recovered subsets are acceptably accurate.

The gross deviation from the MCD subsets is not necessarily, in itself, a bad thing. It was said earlier that all we require of the subsets is that they include all the inliers before any outliers, and they need not always coincide with, nor even be similar to, the MCD for this to happen. What *is* problematic is the negative effect this can have on the test statistics used in hypothesis testing, which is explored further in section 5.3.3.

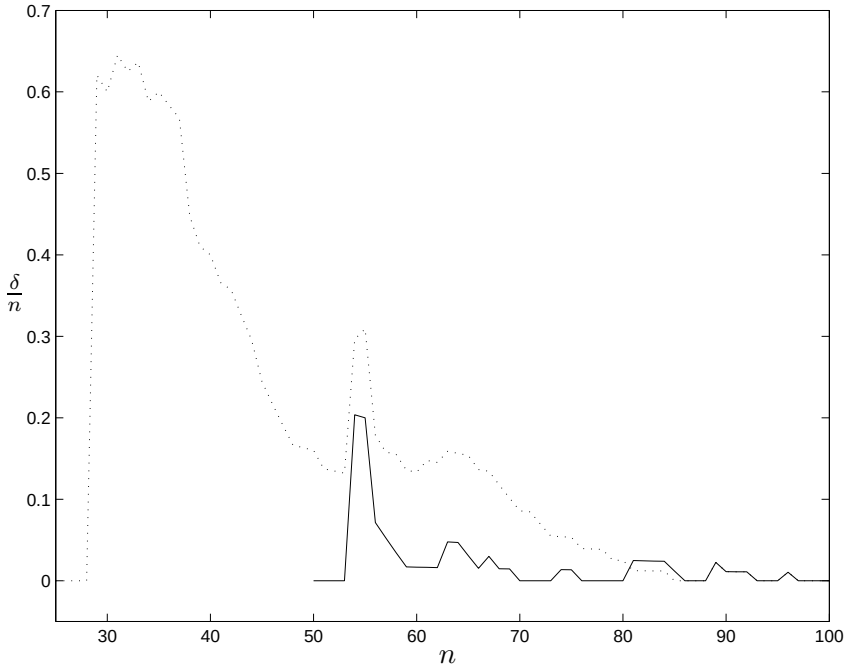


Figure 4.20: Recovery paths from MCD_{25} and MCD_{50}

Chapter 5

Multivariate hypothesis testing

5.1 Introduction

The natural multivariate extension of the univariate test statistic used in chapter 3 is the Mahalanobis distance. That is,

$$z_i = \tfrac{1}{n-1}(\mathbf{x}_i - \bar{\mathbf{x}})^T \mathbf{S}^{-1}(\mathbf{x}_i - \bar{\mathbf{x}}), \tag{5.1}$$

with $z_{(n)}$ used as the test statistic. The fact (see Wilks [28]) that

$$z_i \sim \text{Beta}\left(\frac{p}{2}, \frac{n-p-1}{2}\right), \tag{5.2}$$

makes the use of the Bonferroni inequality and the Independence assumption a straightforward generalization of the univariate case. Wilks [29] used the former to find critical values for a discordancy test of a single outlier. He also considered the distributions of $z_{(n-1)}$, $z_{(n-2)}$, and $z_{(n-3)}$ for block testing of up to four outliers, and tabulated critical values for the testing of one and two outliers.

Caroni and Prescott [6] applied Rosner's simplification of his multiple outlier criterion to the sequential testing of multiple multivariate outliers. They go a little further than Rosner [22] in discussing why this can be expected to be accurate, but avoid the kind of detailed explanation we gave in section 3.1. We feel that our arguments apply equally well to multivariate inlier subsets and their properties, and they are therefore not repeated here. Caroni and Prescott [6] conducted simulations analogous to Rosner's at $\alpha = 5\%$ for $p = 2, 3, 4, 5$ and sample sizes up to 100, also using EDR. They came to the same conclusion that samples of size 25 and greater produce significances close to the nominal one. They remark that this "does not

seem to depend very much on the dimensionality p", but we consider this somewhat naïve, given that only low-dimensional data were tested. We conjecture that it depends on the number of observations over and above the minimum required for a non-trivial estimate of the covariance matrix. That is, a relationship of the form $N \geq p + 24$ is probably pertinent, and it is understandable that this would not be clear from their simulations.

We again choose to focus our efforts on assessing the accuracy of the Bonferroni inequality and the Independence assumption in approximating the null distribution of $z_{(n)}$. We begin by noting that from (5.1) it can be shown (using arguments similar to those in (4.1)) that

$$\sum_{i=1}^{n} z_i = \frac{pn}{n-1}. \tag{5.3}$$

By following the same procedure as in (3.21), we find that an upper bound for $z_{(n-1)}$ is

$$\frac{pn}{2(n-1)}, \tag{5.4}$$

which is greater than one for $p \geq 2$. The Bonferroni inequality is therefore never exact for multivariate $z_{(n)}$, which is further incentive for a simulation study.

5.2 Simulation study

We carried out simulations for dimensions $p = 2, \ldots, 6, 10, 20, 50$ for sample sizes $n = p + 2, \ldots, p + 25$. Many applications of multivariate statistics involve data of only a few dimensions, but the higher dimensions were necessary in order to investigate the asymptotic behaviour of the distribution. For each n, p pair, 10^6 samples from $N(\mathbf{0}, \mathbf{I})$ were generated, with $z_{(n)}$ being recorded for each. A little over 10^{11} standard normal variables were therefore generated, not to mention the additional matrix operations needed to calculate the test statistics. The results of this section are the culmination of several weeks of computing!

As was done in section 3.3, the true significance probabilities of critical values calculated at a nominal $\alpha = 5\%$ using both the Bonferroni inequality and the Independence assumption, were estimated using the simulated data. An inspection of the results showed that, as in the univariate case, these remain more or less constant over the range of n tested. Taking the average significance of both methods is therefore be a good indication of their accuracy for small n. These are shown in Table 5.1, and are illuminating indeed. Using the exact critical values for $p = 1$ as a yardstick

p	Bonferroni	Independence
1	0.0499	0.0511
2	0.0496	0.0508
3	0.0494	0.0505
4	0.0493	0.0505
5	0.0493	0.0504
6	0.0492	0.0503
10	0.0491	0.0503
20	0.0490	0.0502
50	0.0488	0.0500

Table 5.1: Average simulated significance probabilities

for the accuracy of the simulation itself (see section 3.3), it is likely that most of the values in Table 5.1 are correct to all three significant figures given. As expected, those of the Bonferroni inequality are all slightly below 0.05, but they show a steady decrease in accuracy as p increases. The Independence assumption produces significance probabilities *above* 0.05 that *increase* in accuracy with p. For $p \geq 3$ it is more accurate than the Bonferroni inequality, and at a significance of $\alpha = 1\%$ it is better for $p \geq 4$. Although both methods are sufficiently accurate for most applications, it is reassuring that our preferred method has favourable asymptotic properties. Unless otherwise stated, all significance probabilities in the examples that follow are calculated with the Independence assumption.

It is not just the upper tail of the Independence assumption that improves with p. Figure 5.1 shows the histogram of the simulation data for $p = 5$ and $n = p + 5$. The overall fit is considerably better than that of the corresponding sample size for univariate data shown in Figure 3.3. Samples of sizes $n = 50, 100, 200, 500, 1000$ for dimension $p = 5$ were also simulated to investigate the convergence as n increases. This was found to be similar to the slow rate observed in the univariate case.

5.3 Examples

5.3.1 Outlier detection of Lumber data

The data for this example are taken from [15] (p 203), and consist of four measurements (two static and two dynamic) of the stiffness of 30 pieces of lumber. It has been chosen because it is used in [15] as an example in a section titled "Detecting Outliers and Cleaning Data" and is therefore known to contain some outliers. The

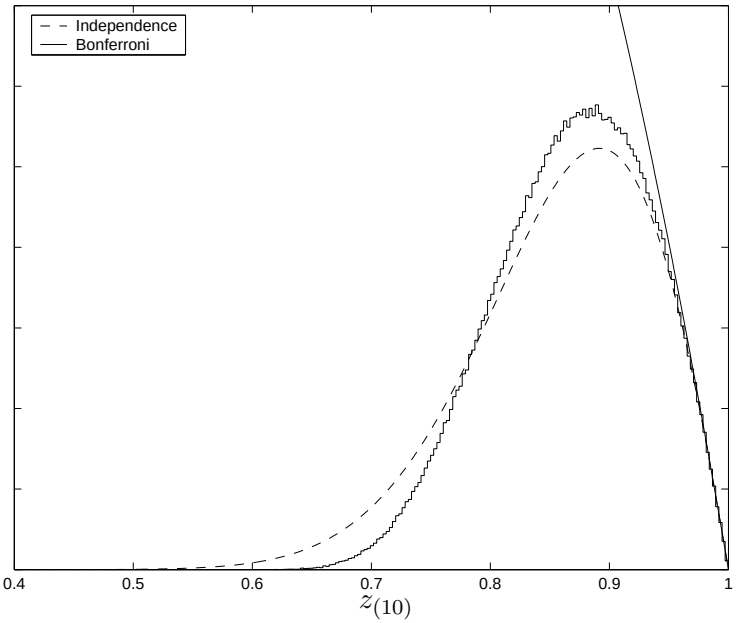


Figure 5.1: Histogram of simulation results for $p = 5$ and $n = 10$

investigation carried out in [15] is rather casual, with scatter plots of the bivariate distributions and a chi-square plot of the Mahalanobis distances being used. A chi-square plot uses abscissae scaled such that a sample which is distributed as chi-square appears linear. They find that two points are outliers, and conclude that "the remaining pattern conforms to the expected straight-line relation", and is therefore free of outliers.

We decided to allow for the worst possible contamination and take $k = 15$. The results are shown in Table 5.2. The index of the observation added to the previous inlier subset to obtain the current one is shown in column two. Column three shows the discrepancy between the recovered subsets and the estimates of the MCD subsets using Fast-MCD with $m = 200$ restarts. Significance probabilities using the Independence assumption, the simulation data (which ends at $n = 29$) and the Bonferroni inequality (where significances above one have been replaced with a –), are all presented. The simulated ones are probably all accurate to the three decimal places given, but all three methods agree that I_{27} is the last inlier subset to pass the hypothesis test at a significance of $\alpha = 5\%$. The remaining three observations 9, 3, and 16 are therefore declared outliers.

Surprisingly, the two outliers identified in [15] are 9 and 16, but according to

n	a	δ	Independence	Simulation	Bonferroni
16	–	0	0.350	0.371	0.424
17	18	0	0.470	0.507	0.623
18	24	0	0.370	0.394	0.456
19	8	2	0.589	0.643	0.869
20	1	1	0.930	0.979	–
21	11	0	0.938	0.983	–
22	19	0	0.947	0.987	–
23	29	0	0.656	0.716	–
24	7	0	0.491	0.528	0.665
25	4	0	0.472	0.507	0.631
26	2	0	0.713	0.777	–
27	21	0	0.223	0.231	0.251
28	9	0	0.029	0.029	0.029
29	3	0	0.018	0.018	0.018
30	16	0	0.003	–	0.003

Table 5.2: Significances of I_n for the Lumber data

our analysis 3 is more aberrant than 9, so if 9 is an outlier, then it follows that 3 and 16 must be too. This highlights the dangers of relying on improvised graphical techniques for detecting outliers.

5.3.2 Cluster Analysis of Crab data

Each observation from this data set is made up of five measurements of the carapace (main shell) of the rock crab *Leptograpsus variegatus*, and is available online [26] in the file `prnn`. Four of the measurements are shown in Figure 5.4, with the fifth one being depth.

The crab appears in two distinct colour forms (orange and blue), but zoologists later decided that these are actually two separate species. This called for a re-examination of a collection of chemically preserved crabs, but because these had lost their distinguishing colour, they had to be classified first. The dataset we use is the training set of the required discriminant analysis, and consists of 50 specimens of each sex and colour. The result is four multivariate normal clusters in close proximity to each other, which is ideal for testing a clustering procedure.

The bivariate scatter plot of Figure 5.2 is one of the more revealing of the $\binom{5}{2} = 10$ possible. It appears to show two clusters that both have a highly correlated positive relationship between the two variables. The plot of Figure 5.3 also shows a strong

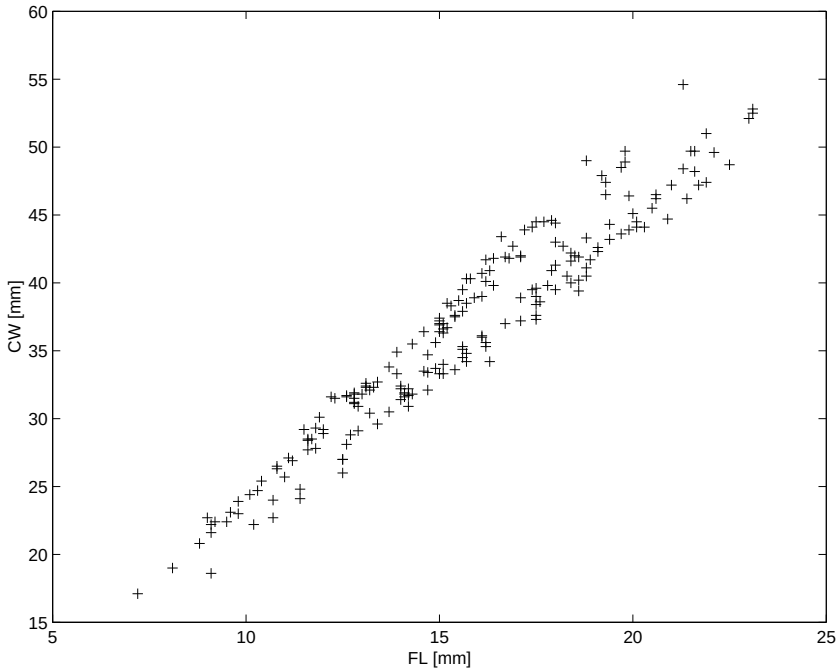


Figure 5.2: Scatter plot of carapace width (CW) and frontal lip (FL)

positive correlation, but there is no sign that anything other than a single cluster is present.

We set $h = 30$ because we know that each cluster contains 50 points. In section 5.3.3 we discuss a method of choosing a suitable h if no *a priori* information exists as to the number and size of the clusters.

Before we attempt to find MCD_{30} we must decide on a reasonable number restarts for Fast-MCD. Equation (4.22) was devised for outlier detection, and requires some modification for cluster analysis. This is because we are not really interested in the true global MCD, but in the MCD of any one of the clusters. This helps to reduce the number of restarts required, since the probability of finding a clean starting subset is increased. The probability of finding a local MCD may be estimated by

$$\Pr\{\text{Local MCD is found}\} = 1 - \left(1 - \rho g \left(\frac{1-\kappa}{g}\right)^{p+1}\right)^{m}, \tag{5.5}$$

where the simplifying assumption that the non-contaminated data is equally divided amongst the g clusters, has been made. This is also conservative, since it minimizes

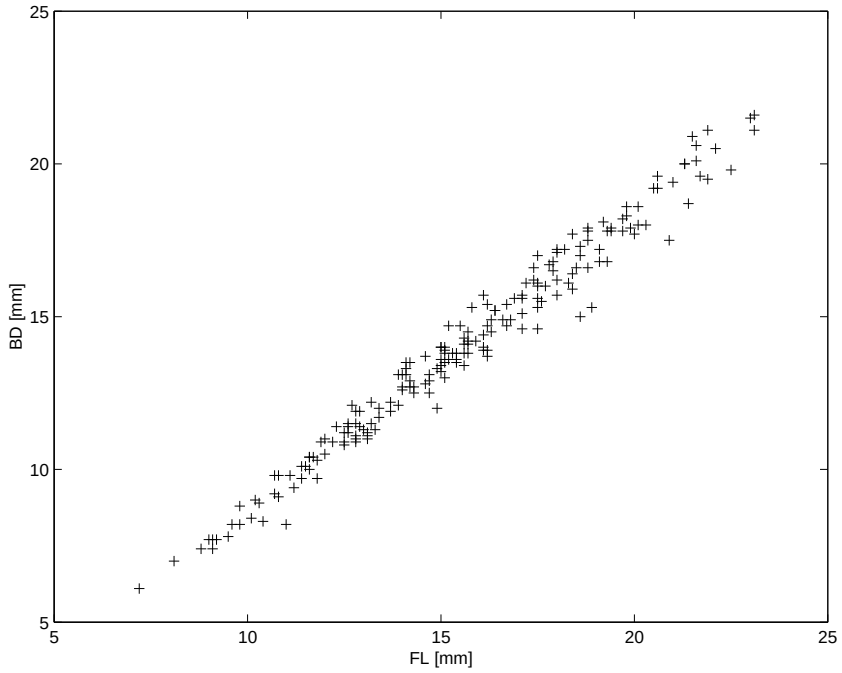


Figure 5.3: Scatter plot of body depth (BD) and frontal lip (FL)

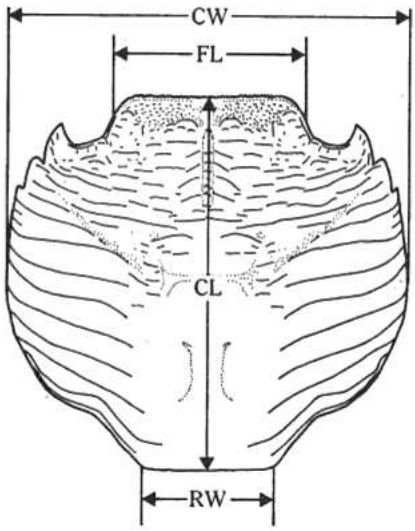


Figure 5.4: Four of the measurements taken of the carapace

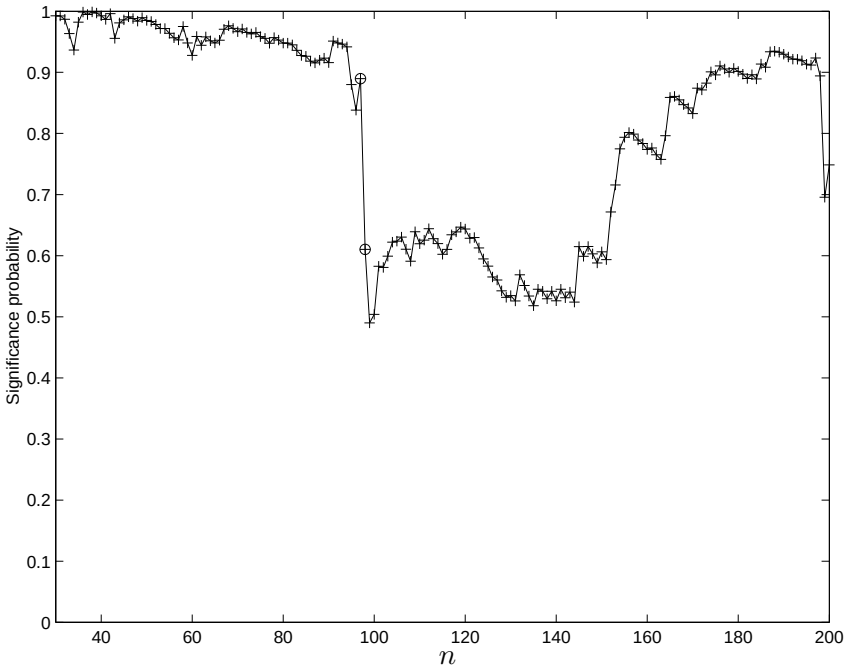


Figure 5.5: First set of significance probabilities

the probability of selecting a clean starting subset, as may be proved using the power mean inequality ([12], p 26).

If we tentatively assume that the global convergence probability is about the same as for bivariate data (in chapter 6 we suggest that this may well be over optimistic), then for $\kappa = 0.1$, $g = 4$, and $m = 10000$, expression (5.5) gives FastMCD a 48% chance of finding a local MCD, which is ample for our purposes.

The significance probabilities of the recovered subsets are shown in Figure 5.5. These are high until a sudden drop occurs from I_{97} to I_{98} (significances circled in Figure 5.5), which is a sure sign that a cluster is on the verge of being completed. Since we know the true number and size of the clusters, we note that unfortunately, two clusters have been merged. We err on the side of caution and put I_{97} aside for the time being, even though it is possible that I_{98} or higher is the complete cluster. It is better to remove an incomplete subset than one which contains observations that do not belong to it.

MCD_{30} was estimated for the remaining points, and the resulting significances shown in Figure 5.6. Another sharp drop occurs, this time from I_{54} to I_{55}. This second cluster is smaller than the first, so the significances tend to be lower and less

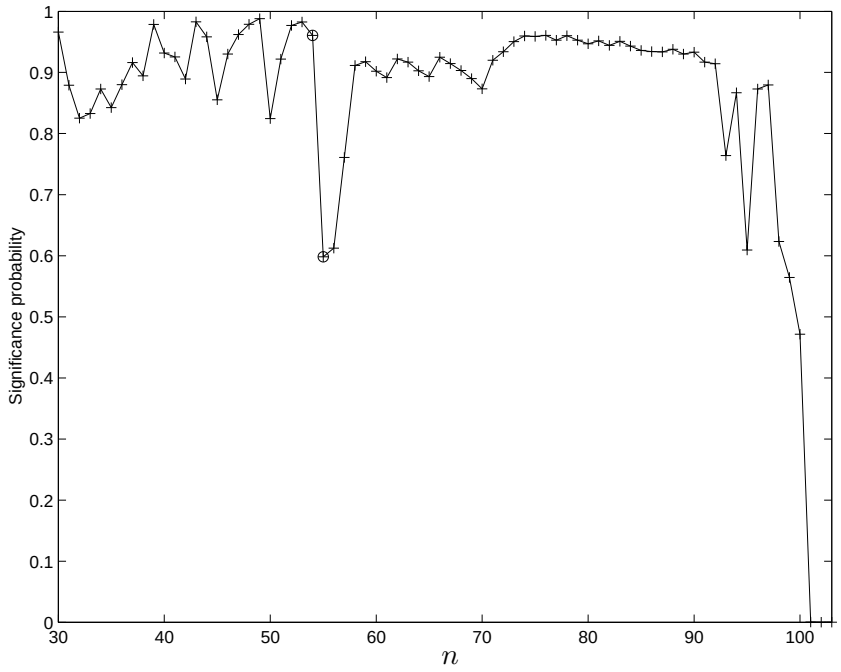


Figure 5.6: Second set of significance probabilities

predictable*. The process was repeated, and a third cluster of at least 40 points was found after examining the plot shown in Figure 5.7.

This preliminary sweep through the data left several points unclassified. Each of these must either be assigned to one of the clusters or be declared an outlier. Our algorithm calculates the significance probability of the first point that would be added if recovery took place from each of the existing clusters. The cluster with the largest significance has the corresponding point added to it. It is possible that two or more clusters will sometimes be competing for the same point. When less than g points remain, this will *always* be the case, so we term this assignment the *arbitration* phase. If the maximum of the g significance probabilities drops below α, then it is terminated, and all the remaining observations are declared outliers.

Figure 5.8 shows a discriminant space plot of the data. This can be thought of as a projection of the data that maximizes the separation of the groups by using the first and second of Fisher's sample linear discriminants [15] (p 683 ff). This was achieved using the four known clusters, not the results of our analysis.

The first cluster we detected was increased to size 100 during arbitration and

*This behaviour is also seen in Table 5.2

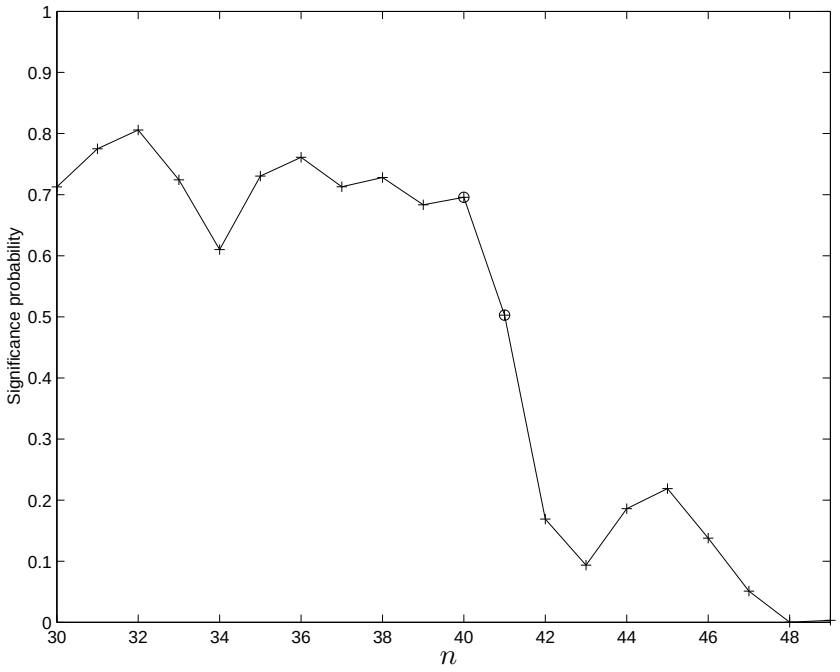


Figure 5.7: Third set of significance probabilities

consists of all the specimens from the blue species. The fact that we were unable to distinguish between sex is understandable considering the overlap of the two clusters seen in Figure 5.8. The second cluster had one more point added to it, and consists of all 50 orange males as well as 5 orange females. The five incorrectly assigned points are circled in Figure 5.8. The last cluster is made up the remaining 45 orange females, since no outliers were found using $\alpha = 5\%$.

Generally speaking, our analysis is satisfactory, especially since the authors of [16] stress that this data set "is quite difficult for most clustering algorithms." In addition, no attempt is made in [16] to distinguish between sex, whereas our analysis separated the orange specimens fairly accurately, and with comparative ease.

5.3.3 Artificial Normal data

We return to the data of section 4.10 to complete our investigation into the properties of the recovered subsets by considering their significance probabilities. These are shown in Figure 5.9, where it is seen that those produced from the subsets recovered from MCD_{25} are substantially lower than the others. This could be confusing, and might lead the analyst to suspect that the data are not multivariate normal.

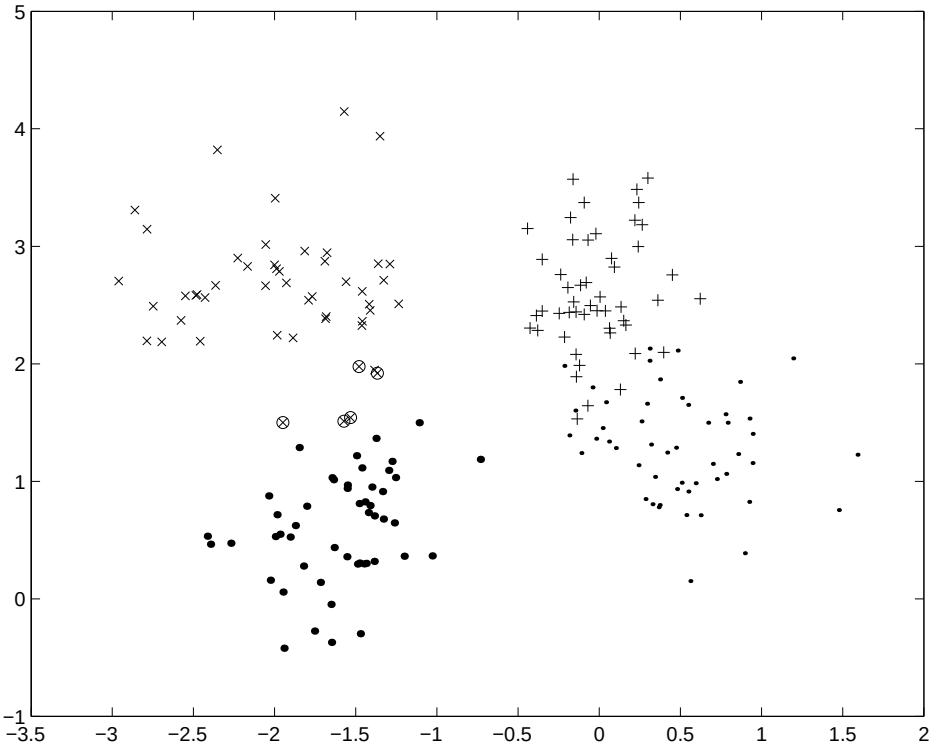


Figure 5.8: Discriminant space plot. Orange females (×), orange males (•), blue females (+), and blue males (·).

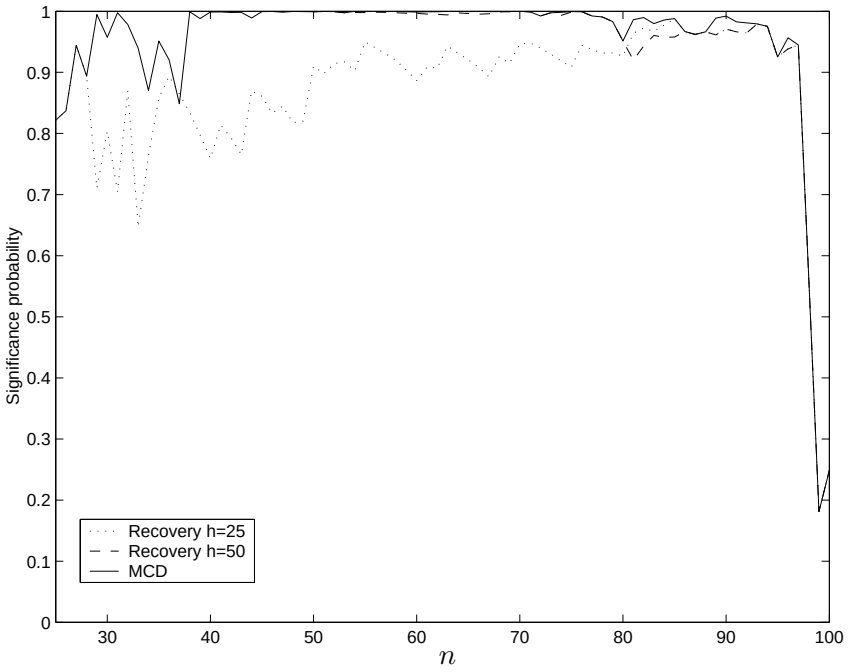


Figure 5.9: Significance probability paths of three sets of I_n

Because it indicates that the seed subset is becoming unstable, this characteristic can also be used to our advantage. Smaller h will not be beneficial and may even be dangerous (in that recovery could include outliers before inliers), so finding this threshold is useful for choosing h in a cluster analysis. We could start with $h = \lfloor N/2 \rfloor$ and decrease it (perhaps by a factor of 2), until the significance probabilities start out low and deviate markedly from the paths taken by the other recovered subsets.

For this dataset, $h = 38$ is the required threshold for stable recovery. This can be seen from Figure 5.9 by the consistently high significance probabilities of the MCD subsets of sizes $n \geq 38$. All subsets recovered from seed subset sizes $25 \leq h < 38$ arrive at one of two I_{37} subsets and therefore follow either the path shown in Figure 5.9 or another, similarly erratic one, from there on. In contrast, all those recovered from seed subset sizes $h \geq 38$ follow paths so close to those of the MCD subsets and the subsets recovered from MCD_{50}, that they would be barely visible in the plot of Figure 5.9.

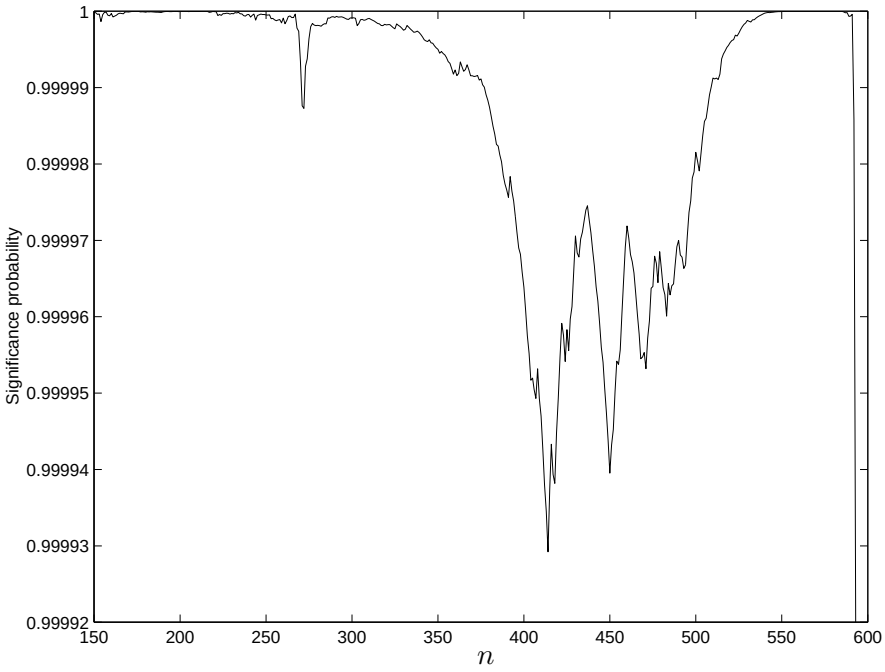


Figure 5.10: Significance probabilities of artificial clusters

5.3.4 Artificial Normal clusters

This section looks at when to begin the arbitration phase by performing a cluster analysis on a bivariate data set composed of two clusters each of size 300. One is $N(\mathbf{0}, \mathbf{I})$ and the other $N([4, 0]^T, \mathbf{I})$. The resulting significance probabilities using $h = 150$ and $m = 100$ are shown in Figure 5.10. As can be seen from the ordinate range of 0.99992 to 1, these are very high as a result of the large number of observations.

For the crab data it was decided that it would not be worth trying to find a fourth cluster with the remaining 9 points. From the very slight yet tell-tale drop observed in Figure 5.10 at about $n = 270$, we infer that the first cluster is very near to the other, and that many points still need to be assigned to it. We might otherwise have identified a spurious extra cluster from the 55 points that remained after identifying the core of each of the two that are known to exist. Figure 5.11 is a scatter plot of the data, where observations from the first identified cluster are circled. No outliers were found for $\alpha = 5\%$, so all the other observations form the second cluster. It is evident from this example just how well the arbitration phase works. The overlap naturally causes a number of observations to be incorrectly assigned, but it is otherwise hard to find fault with the final categorization.

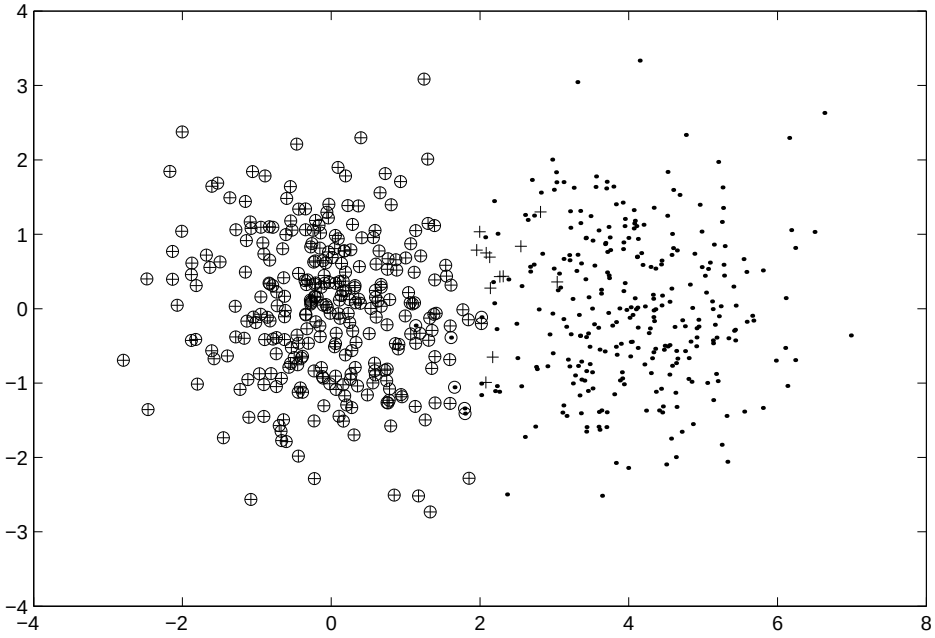


Figure 5.11: Final result of the cluster analysis

Chapter 6

Conclusions

The reliable detection of multiple multivariate outliers cannot be achieved using ad hoc methods. Only methods that combine a carefully designed algorithm for finding inlier subsets, and a means of carrying out meaningful hypothesis testing on them, seems capable of delivering consistently good results. Even then, there is no substitute for thorough testing on simulated data, and this principle was used throughout this thesis.

Our simulation study of the null distribution of $z_{(n)}$ revealed that while the accuracy of both the Bonferroni inequality and the Independence assumption are adequate for most problems, the latter exhibits favourable asymptotic properties that converge to the exact distribution as both p and n increase, whereas the former gradually worsens. At a significance of 5%, the Independence assumption is the more accurate of the two for $p \geq 3$, and nearly as good for uni- and bivariate test statistics, so it is not difficult to recommend it as a replacement for the Bonferroni inequality. Another advantage is that its reasonable accuracy over the full range of $z_{(n)}$ makes it possible to separate overlapping clusters.

Our simulations strongly suggest (see Tables 5.1 and 5.2) that the Independence assumption yields a lower bound for the true significance. Proving this would be a very useful theoretical result, because in conjunction with the Bonferroni inequality, it would provide one with (for small significance probabilities at least) a narrow interval in which the true significance lies.

The multivariate inlier subsets revolve around the MCD, but the fastest algorithm known (MCD-BF of section 4.5) is $O(N^{(p+1)(p+2)/2})$. The order of complexity grows quadratically with p, so it becomes very expensive for high dimensional data. This poses an interesting question regarding the computation of the MCD. Does an

N	$\hat{\rho}$	N	$\hat{\rho}$	N	$\hat{\rho}$	N	$\hat{\rho}$
10	0.369	50	0.432	90	0.283	130	0.236
20	0.305	60	0.301	100	0.289	140	0.269
30	0.304	70	0.362	110	0.346	150	0.389
40	0.446	80	0.415	120	0.407	160	0.442

Table 6.1: Estimates of ρ for univariate data

algorithm exist whose order of complexity grows linearly with p?

For example, if all $\binom{N}{p+1}$ possible starting subsets are used for Fast-MCD, then it seems extremely unlikely that it will fail to find the MCD. Might it be possible to prove that such a search *must* find the MCD? If so, and assuming that the average number of iterations until convergence doesn't grow too fast (see section 4.8.2), then this would be such an algorithm. It would be very slow however, and it might still be better just to use Fast-MCD with a fraction of all the possible starting subsets that finds the MCD with high probability.

This brings us to the question of the reliability of Fast-MCD, and our definition of ρ, the global convergence probability. A study of its value for dimensions higher than two is crucial in assessing the usefulness of Fast-MCD for such data. In order to gain further insight into how ρ might vary with p, we modified our Fast-MCD code to allow for univariate data. The univariate MCD is of course MVAR, which is easily obtained (see section 2.1). Simulations identical to those that produced Table 4.1 were repeated for univariate data, and the results are given in Table 6.1. It is immediately apparent that considerably higher values have been found, with the average of 0.350 nearly three times that of bivariate data. Although there is again no evidence that ρ changes with N, it is rather disconcerting that it might continue to drop for higher dimensions. Does it reach some limiting value? What if this limit is zero?

Exact algorithms for $p > 2$ are clearly needed. The obvious choice is MCD-BF, but as we have already pointed out, it becomes very expensive for $p > 2$, and is probably limited to $p = 3$. The geometric algorithms MCD-SW and MCD-EXT could also be implemented for three dimensions, although this would not be easy, and higher dimensions are guaranteed to be exceedingly difficult. It may turn out that MCD-BB, which we were quick to dismiss in section 4.6.2 because of its exponential complexity, is the only feasible algorithm for dimensions greater than three. The apparently constant nature of ρ for a given dimension could be exploited by testing only smaller samples, which is further reason to believe that MCD-BB

will be indispensable for future research.

Until more is known about ρ for higher dimensions, we should pessimistically assume that it does decrease. Even if this is not by much, the number of restarts required to be confident of finding the MCD will still be very high if κ is large, because the probability of starting with a clean subset of size $p+1$ decreases exponentially. This becomes computationally formidable if a number of clusters are presumed to be present, and it is vital that the most is made of the resulting MCD estimate. The recovery procedure (section 4.9.2) fills this role almost perfectly, with the only limitation being the threshold of stability discussed in section 5.3.3. It is essential that the analyst experiment with simulated data so as to develop a feel for it before using the technique on real data. The same can be said of estimating the extent of overlap amongst clusters. Both of these would have to be studied further and quantified if the entire clustering procedure is to be automated. We doubt if this could be done satisfactorily without inordinate effort, and recommend that it be used in the supervised manner as applied in the examples.

The MCD is not even mentioned in the practical pattern recognition texts [4] and [9], as these appeared before Fast-MCD was developed. We believe that it is only a matter of time before Fast-MCD, along with our techniques presented in this thesis, find useful application in this and other fields whose foundation is the multivariate normal distribution.

Bibliography

[1] Banfield, J. D., and Raftery, A. E. (1993), Model-Based Gaussian and Non-Gaussian Clustering, *Biometrics*, **49**, 803-821.

[2] Barnett, V., and Lewis, T. (1994), *Outliers in Statistical Data* - 3rd edition, John Wiley & Sons, Chichester.

[3] Bernholt, T., and Fischer, P. (2001), The Complexity of Computing the MCD-Estimator, Unpublished paper available online at
`http://ls2-www.cs.uni-dortmund.de/~bernholt/`

[4] Bishop, Christopher M. (1995), *Neural Networks for Pattern Recognition*, Clarendon Press, Oxford.

[5] Bogart, Kenneth P. (1983), *Introductory Combinatorics*, Pitman Publishing Inc., Boston.

[6] Caroni, C., and Prescott, P. (1992), Sequential Application of Wilks's Multivariate outlier test, *Applied Statistics*, **41**, 355-364.

[7] Celeux, G., and Govaert, G. (1993), Comparison of the Mixture and the Classification Maximum Likelihood in Cluster Analysis, *Journal of Statistics, Computation, and Simulation*, **47**, 127-146.

[8] de Berg, M., van Kreveld, M., Overmars, M., and Schwarzkopf, O. (1997), *Computational Geometry - Algorithms and Applications*, Springer.

[9] Fukunaga, K. (1990), *Introduction to Statistical Pattern Recognition*, Academic Press, Inc.

[10] Ganesalingam, S. (1989), Classification and Mixture Approaches to Clustering via Maximum Likelihood, *Applied Statistics*, **38**, No. 3, 455-466.

[11] Gnanadesikan, R., and Kettenring, J. R. (1972) Robust Estimates, Residuals, and Outlier Detection with Multiresponse Data, *Biometrics*, **28**, 81-124.

[12] Hardy, G. H., Littlewood, J. E., and Pólya, G. (1967) *Inequalities*, Cambridge University Press, Cambridge.

[13] Hawkins, D. M. (1980), *Identification of Outliers*, Chapman & Hall, London.

[14] Hawkins, D. M. (1994), The feasible solution algorithm for the minimum covariance determinant estimator in multivariate data, *Computational Statistics & Data Analysis*, **17**, 197-210.

[15] Johnson, R. A., and Wichern, D. W. (1998), *Applied Multivariate Statistical Analysis* - 4th edition, Prentice Hall, New Jersey.

[16] Kamvar, S. D., Klein, D., and Manning, C. D. (2002), Interpreting and Extending Classical Agglomerative Clustering Algorithms using a Model-Based Approach *ICML*, 283-290. Available online at `http://www.stanford.edu/~danklein/`

[17] Kaufman, L., and Rousseeuw, P. J. (1990), *Finding Groups in Data - An Introduction to Cluster Analysis*, John Wiley & Sons, New York.

[18] Metcalfe, Andrew V. (1994), *Statistics in Engineering - A practical approach*, Chapman & Hall, London.

[19] Pesch, C. (1998), Fast Computation of the Minimum Covariance Determinant Estimator, *Technical Report MIP-9806*, Fakultät für Mathematik und Informatik, Universität Passau,
`http://stoch.fmi.uni-passau.de/publikationen/MIP-9806.ps`

[20] Pesch, C. (2000), Eigenschaften des gegenüber Ausreißern robusten MCD-Schätzers und Algorithmen zu seiner Berechnung, Ph.D. thesis, Universität Passau.

[21] Rocke, David M., and Woodruff, David L. (1998), Some Statistical Tools for Data Mining Applications, Unpublished paper available online at
`http://handel.cipic.ucdavis.edu/~dmrocke/preprints.html`

[22] Rosner, B. (1983), Percentage points for a generalized ESD many-outlier procedure, *Technometrics*, **25**, 165-172.

[23] Rousseeuw, P. J. (1984), Least Median of Squares Regression, *Journal of the American Statistical Association*, **79**, 871-880.

[24] Rousseeuw, P. J., and van Driessen, K. (1999), A Fast Algorithm for the Minimum Covariance Determinant Estimator, *Technometrics*, **41**, Number 3, 212-223.

[25] Rousseeuw, P. J., and Leroy, A. M. (1987), *Robust Regression and Outlier Detection*, John Wiley & Sons, New York.

[26] *Statlib Datasets Archive*, hosted by Carnegie Mellon University at `http://lib.stat.cmu.edu/datasets/`

[27] Viljoen, H. (1999), A computer intensive approach to identify multiple outliers, Ph.D. thesis, Potchefstroom University.

[28] Wilks, S.S. (1962), *Mathematical Statistics*, John Wiley & Sons, New York.

[29] Wilks, S.S. (1963), Multivariate statistical outliers, *Sankhyâ, Series A*, **25**, 407-426.

Appendix A

Mathematical proofs and notes

A.1 Stirling's number of the second kind

The number of partitions of N distinct objects into g non-empty groups, is commonly denoted by $S(N, g)$. The recursive relationship

$$S(N, g) = S(N-1, g-1) + gS(N-1, g), \tag{A.1}$$

is easily proved ([5], p 48). Because empty groups are not allowed, $N \geq g$. Using the fact that $S(N, 1)$ and $S(N, N)$ both equal one allows for the computation of any $S(N, g)$, although the expression

$$S(N, g) = \frac{1}{g!} \sum_{i=0}^{g} (-1)^i \binom{g}{i} (g-i)^N, \tag{A.2}$$

which may be verified using (A.1), is more direct.

A.2 Student's t and the Beta distribution

It is interesting to note that MATLAB calculates the percentiles of the t distribution using the following relationship with the Beta distribution. We wish to show that the random variable

$$y = \frac{t}{\sqrt{v + t^2}}, \tag{A.3}$$

where t is a Student's t random variable with v degrees of freedom, is equivalent to

$$y = 2x - 1, \tag{A.4}$$

where x is Beta$\left(\frac{v}{2}, \frac{v}{2}\right)$. From (A.3) we obtain

$$t = \frac{\sqrt{v}y}{\sqrt{1-y^2}}, \tag{A.5}$$

and therefore

$$\frac{\partial t}{\partial y} = \frac{\sqrt{v}}{(1-y^2)^{\frac{3}{2}}}. \tag{A.6}$$

The density for y is given by

$$\begin{aligned} f_y(y) &= f_t(t(y)) \left|\frac{\partial t}{\partial y}\right| \\ &= \frac{\Gamma\left(\frac{v+1}{2}\right)}{\Gamma\left(\frac{v}{2}\right)\sqrt{v\pi}} \left(1 + \frac{vy^2}{(1-y^2)v}\right)^{-\left(\frac{v+1}{2}\right)} \frac{\sqrt{v}}{(1-y^2)^{\frac{3}{2}}} \\ &= \frac{\Gamma\left(\frac{v+1}{2}\right)}{\Gamma\left(\frac{v}{2}\right)\sqrt{\pi}} \left(1-y^2\right)^{\frac{v}{2}-1}. \end{aligned} \tag{A.7}$$

The partial derivative is always positive, so the absolute value is removed. Following the same procedure for the simple linear transformation given in (A.4), we arrive at

$$\frac{\Gamma(v)}{2\left(\Gamma\left(\frac{v}{2}\right)\right)^2} \left(\frac{1-y^2}{4}\right)^{\frac{v}{2}-1}, \tag{A.8}$$

which is not obviously equal to (A.7). Ignoring the terms common to both, it is seen that in order to prove equality we must show that

$$\Gamma(v)\sqrt{\pi} = \Gamma\left(\frac{v+1}{2}\right)\Gamma\left(\frac{v}{2}\right)2^{v-1}, \quad \forall v = \{1, 2, \ldots\}. \tag{A.9}$$

For the case $v = 1$, both sides equal $\sqrt{\pi}$. For $v = k+1$ the RHS of (A.9) is

$$\Gamma\left(\frac{k+2}{2}\right)\Gamma\left(\frac{k+1}{2}\right)2^k = \frac{k}{2}\Gamma\left(\frac{k}{2}\right)\Gamma\left(\frac{k+1}{2}\right)2^{k-1}2 \tag{A.10}$$

and using the induction assumption, this becomes

$$k\Gamma(k)\sqrt{\pi} = \Gamma(k+1)\sqrt{\pi}, \tag{A.11}$$

which is the LHS of (A.9) with $v = k+1$.

Appendix B

Computer source code

B.1 Introduction

The four exact MCD algorithms were programmed in C to produce functions that can be accessed from within MATLAB. This allows for the convenience of using MATLAB for generating, sorting, and plotting data; with the speed and flexibility of the compiled C language. MATLAB refer to these external C modules as MEX-files, so we name our source files with the suffix `_mex.c`. Once the `mex` script is set up, compilation from within the MATLAB environment is done with the command `>> mex mcd_bb_mex.c`, which produces a shared object file of the same name with an extension that is system dependent*. An m-script of the same name but without the suffix contains the usage of the command when `>> help mvar` is typed, and also carries out error checking and pre- and post-processing of the data.

B.2 MCD-BB

Contents of `mcd_bb.m`:

```
%
% The Branch and Bound algorithm for the finding the MCD subset.
%
% mcd_indices = mcd_bb(X, n);
%
% where X is the 2 by N data matrix, n is the subset size, and mcd_indices
% is an array containing the sorted indices of the MCD subset.
%

function mcd_indices = mcd_bb(X, n)
```

*For MS Windows this is `dll`, and for the GNU/Linux systems we used it is `mexglx`

```
[ p, N ] = size(X);

if p ~= 2
error 'This function is for bivariate data only'
end

mcd_indices = mcd_bb_mex(X, n);
```

Contents of `mcd_bb_mex.c`:

```
/*
 * Implementation of Pesch's Branch and Bound exact MCD algorithm
 *
 * Started: 21 April 2002
 */

#include "mex.h"
#include <stdio.h>
#include <stdlib.h>
#include <math.h>       /* HUGE_VAL */
#include <string.h>     /* memcpy() */

#define p 2

/* macros to allow for {1,...,n} indices */
#define I(l) (i[l - 1])
#define s_11(l) (Level[l - 1].s_11)
#define s_12(l) (Level[l - 1].s_12)
#define s_22(l) (Level[l - 1].s_22)
#define sumx1(l) (Level[l - 1].sumx1)
#define sumx2(l) (Level[l - 1].sumx2)
#define Det(l) (s_11(l)*s_22(l) - s_12(l)*s_12(l))
#define Answer(j) (answer[j - 1])

unsigned int *i;
unsigned int N, l, n;
struct sums {
    double s_11, s_12, s_22, sumx1, sumx2;
} *Level;
double *X;
double *answer;
double *nodes;
double min_det;
unsigned int j;

void add_point(void);
void add_first_point(void);
int next_subset(void);

void mcd_bb(void)
{
    i = malloc(n*sizeof(unsigned int));
```

```
    Level = (struct sums *) malloc(n*sizeof(struct sums));

    for (j = 1; j <= n; j++)
        I(j) = j;

    add_first_point();

    do {
        while (l < p + 1)
            add_point();

        while (1)
        {
            if (Det(l) > min_det)
                break;
            if (l == n) {
                min_det = Det(l);
                for (j = 1; j <= n; j++)
                    Answer(j) = (double) I(j);
                break;
            }
            add_point();
        }
    } while (next_subset());

    free(i);
    free(Level);
}

int next_subset(void)
{
    l--;
    while (++I(l + 1) > N - n + l + 1)
    {
        if (l-- == 0)
            return 0;
    }

    for (j = l + 2; j <= n; j++)
        I(j) = I(j - 1) + 1;

    if (l == 0)
        add_first_point();
    else
        add_point();
    return 1;
}

void add_first_point(void)
{
    double register x1, x2;
    double *xptr;

    (*nodes)++;
```

```
    xptr = X + (I(1) - 1)*p;
    x1 = *xptr++; x2 = *xptr;

    s_11(1) = 0;
    s_12(1) = 0;
    s_22(1) = 0;
    sumx1(1) = x1;
    sumx2(1) = x2;
    l = 1;
}

void add_point(void)
{
    double register x1, x2, temp1, temp2, temp3;
    double *xptr;

    (*nodes)++;

    xptr = X + (I(l + 1) - 1)*p;
    x1 = *xptr++; x2 = *xptr;
    temp1 = sumx1(l) - l*x1;
    temp2 = sumx2(l) - l*x2;
    temp3 = l*(l + 1);
    s_11(l + 1) = s_11(l) + temp1*temp1/temp3;
    s_12(l + 1) = s_12(l) + temp1*temp2/temp3;
    s_22(l + 1) = s_22(l) + temp2*temp2/temp3;
    sumx1(l + 1) = sumx1(l) + x1;
    sumx2(l + 1) = sumx2(l) + x2;
    l++;
}

void mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[])
{
    /* get mandatory arguments */
    N = mxGetN(prhs[0]);
    X = mxGetPr(prhs[0]);
    n = mxGetScalar(prhs[1]);

    /* allocate memory for mandatory answer */
    plhs[0] = mxCreateDoubleMatrix(1, n, mxREAL);
    answer = mxGetPr(plhs[0]);

    /* optional arguments of a low |B_n| and the subset that produced it */
    if (nrhs == 4) {
        min_det = mxGetScalar(prhs[2]);
        memcpy(answer, mxGetPr(prhs[3]), sizeof(double)*n);
    } else {
        min_det = HUGE_VAL;
    }

    /* optional return of the number of nodes */
    if (nlhs == 2) {
        plhs[1] = mxCreateDoubleMatrix(1, 1, mxREAL);
```

```
        nodes = mxGetPr(plhs[1]);
        *nodes = 0;
    } else {
        /* simplify the rest of the code by giving the pointer memory */
        nodes = malloc(sizeof(double));
    }

    mcd_bb();
}
```

B.3 MCD-SW

Contents of `mcd_sw.m`:

```
%
% The Sweepline algorithm for finding the MCD subset.
%
% mcd_indices = mcd_sw(X, n);
%
% where X is the 2 by N data matrix, n is the subset size, and mcd_indices
% is an array containing the sorted indices of the MCD subset.
%

function mcd_indices = mcd_sw(X, n)

[ p, N ] = size(X);

if p ~= 2
error 'This function is for bivariate data only'
end

X = [ X; 1:N ];
X = sortrows(X')';
original_indices = X(3,:);
X = X([1, 2], :);

mcd_indices = mcd_sw_mex(X, n);
mcd_indices = sort(original_indices(mcd_indices));
```

Contents of `mcd_sw_mex.c`:

```
/*
 * Implementation of Pesch's Sweepline MCD algorithm.
 *
 * Started: 10 September 2002
 * Last modified: 3 November 2002
 */

#include <stdio.h>
#include <stdlib.h>
```

```
#include <string.h>
#include <math.h>
#include "mex.h"

#define CW 1
#define OK 0
#define INVALID 1
#define COMPLETE 2
#define TRUE 0
#define FALSE 1

/* macros to allow for {1,...,n} indices */
#define I(l) (i[l - 1])
#define Selected(i) (selected[i - 1])
#define Answer(j) (answer[j - 1])
#define Min(a,b) ((a > b)? b:a)

double *answer, *complete, *nodes, *X;
int N, n;

struct point {
    int i;
    double x, y;
    struct point *next, *previous;
    struct point *segment_start, *segment_end, *old_current;
};

struct point *current;
int *i, *selected, l, j;

struct point* new_point(int i);
int on_right(struct point *origin, struct point *dest, int index);
void restore_convex_hull(void);
void add_first_point(void);
void add_second_point(void);
int check_point_and_add(void);
int next_subset(void);
double calculate_det(void);

void mcd_sw(void)
{
    double min_det = HUGE_VAL, new_det;

    i = malloc(n*sizeof(int));
    selected = malloc(N*sizeof(int));
    memset(selected, 0, N*sizeof(int));

    for (j = 1; j <= n; j++)
        I(j) = j;

    add_first_point();
    add_second_point();

    do {
```

```
        /* continue building up convex hull until either a foreign point
         * is included or until subset is complete. Note: If the former,
         * then the new point is NOT added to the convex hull */
        while (check_point_and_add() == OK);
        if (l == n)
        {
            (*complete)++;
            /* convex complete set found */
            new_det = calculate_det();
            if (new_det < min_det) {
                min_det = new_det;
                for (j = 1; j <= n; j++)
                    Answer(j) = (double) I(j);
            }
            restore_convex_hull();
        }
    } while (next_subset() == TRUE);

    free(i);
    free(selected);
}

/*
 * Create a new subset
 */
int next_subset(void)
{
    while (++I(l + 1) > N - n + l + 1)
    {
        if (l == 0)
            return FALSE;
        restore_convex_hull();
    }

    for (j = l + 2; j <= n; j++)
        I(j) = I(j - 1) + 1;

    switch (l) {
        case 0: add_first_point();
        case 1: add_second_point();
    }
    return TRUE;
}

/*
 * Build up the convex hull and test the new region
 */
int check_point_and_add(void)
{
    double tan_start, tan, tan_before, x, y;
    struct point *test, *search, *upper, *lower;

    /* create new point */
    test = new_point(I(l + 1));
```

```
(*nodes)++;

tan_start = (current->y - test->y)/(test->x - current->x);

/* find the upper tangent */
tan = tan_start;
search = current;
do {
    tan_before = tan;
    search = search->previous;
    tan = (search->y - test->y)/(test->x - search->x);
} while (tan > tan_before);
upper = search->next;

/* find the lower tangent */
tan = tan_start;
search = current;
do {
    tan_before = tan;
    search = search->next;
    tan = (search->y - test->y)/(test->x - search->x);
} while (tan < tan_before);
lower = search->previous;

/* find any foreign points */
for (j = Min(lower->i, upper->i) + 1; j <= I(l+1)-1; j++) {
    if (!Selected(j))
    {
        x = *(X + (j - 1)*2);
        y = *(X + (j - 1)*2 + 1);
        if (((x - upper->x)*(test->y - upper->y)
            > (y - upper->y)*(test->x - upper->x))
            &&
            ((x - test->x)*(lower->y - test->y)
            > (y - test->y)*(lower->x - test->x))
            &&
            ((x - lower->x)*(upper->y - lower->y)
            > (y - lower->y)*(upper->x - lower->x)))
        {
            free(test);
            return INVALID;
        }
    }
}

/* Abort criterion not reached, so "test" point is
 * valid, and becomes the current point. */
test->old_current = current;
current = test;

/* store pointers to the soon to be discarded segment */
current->segment_start = upper->next;
current->segment_end = lower->previous;
```

```
    /* insert added point into chain */
    current->next = lower;
    upper->next = current;
    current->previous = upper;
    lower->previous = current;

    /* mark point as selected */
    Selected(I(l + 1)) = 1;

    /* check if the subset is complete */
    if (++l == n)
        return COMPLETE;

    return OK;
}

void add_first_point(void)
{
    current = new_point(I(1));
    Selected(I(1)) = 1;
    (*nodes)++;
}

void add_second_point(void)
{
    struct point *second;

    second = new_point(I(2));
    Selected(I(2)) = 1;
    (*nodes)++;

    current->next = second;
    current->previous = second;
    second->next = current;
    second->previous = current;

    current = second;
    l = 2;
}

/*
 * Test if a point lies on the right of a directed line
 */
int on_right(struct point *origin, struct point *dest, int index)
{
    double *xptr = X + (index - 1)*2, x, y;

    x = *xptr++; y = *xptr;
    if ((x - origin->x)*(dest->y - origin->y)
        > (y - origin->y)*(dest->x - origin->x))
        return 1; /* true */
    else
        return 0; /* false */
}
```

```
/*
 * Restore the convex hull to that of the previous level
 */
void restore_convex_hull(void)
{
    struct point *temp = current;

    if (l >= 3) {
        temp = current;
        temp->previous->next = temp->segment_start;
        temp->next->previous = temp->segment_end;
        current = temp->old_current;
        Selected(temp->i) = 0;
        free(temp);
    } else if (l == 2) {
        current = current->previous;
        Selected(current->next->i) = 0;
        free(current->next);
    } else {
        Selected(current->i) = 0;
        free(current);
    }
    l--;
}

/*
 * Dynamically create a new extreme point
 */
struct point* new_point(int i)
{
    struct point* new;
    double *xptr;

    if ((new = malloc(sizeof(struct point))) == NULL) {
        fprintf(stderr, "malloc() error!\n");
        exit(1);
    }

    xptr = X + (i - 1)*2;
    new->x = *xptr++;
    new->y = *xptr;
    new->i = i;

    return new;
}

/*
 * Calculate the determinant of the covariance matrix of the current subset
 */
double calculate_det(void)
{
    double *xptr, x, y;
    double sumx2 = 0,
```

```
        sumxy = 0,
        sumy2 = 0,
        sumx = 0,
        sumy = 0;
    double register temp;

    for (j = 1; j <= n; j++)
    {
        xptr = X + (I(j) - 1)*2;
        x = *xptr++; y = *xptr;
        sumx2 += x*x;
        sumxy += x*y;
        sumy2 += y*y;
        sumx += x;
        sumy += y;
    }
    temp = sumxy - sumy*sumx/n;
    return (sumx2 - sumx*sumx/n)*(sumy2 - sumy*sumy/n) - temp*temp;
}
/*
 * Gateway routine to MATLAB
 */
void mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[])
{
    N = mxGetN(prhs[0]);
    X = mxGetPr(prhs[0]);
    n = mxGetScalar(prhs[1]);

    plhs[0] = mxCreateDoubleMatrix(1, n, mxREAL);
    answer = mxGetPr(plhs[0]);

    /* Three arguments to return, the indices, the number of convex complete
     * subsets, and the nodes */
    if (nlhs == 3) {
        plhs[1] = mxCreateDoubleMatrix(1, 1, mxREAL);
        nodes = mxGetPr(plhs[1]);
        *nodes = 0;

        plhs[2] = mxCreateDoubleMatrix(1, 1, mxREAL);
        complete = mxGetPr(plhs[2]);
        *complete = 0;
    } else {
        /* simplify the rest of the code by giving the pointers memory */
        nodes = malloc(sizeof(double));
        complete = malloc(sizeof(double));
    }

    /* call the actual routine */
    mcd_sw();
}
```

B.4 MCD-EXT

Contents of `mcd_ext.m`:

```
%
% The Extreme points algorithm for finding the MCD subset.
% The simplest usage is:
%
% mcd_indices = mcd_ext(X, n);
%
% where X is the 2 by N data matrix, n is the subset size, and mcd_indices
% is an array containing the ordered indices of the MCD subset.
%

function mcd_indices = mcd_ext(X, n)

[ p, N ] = size(X);

if p ~= 2
error 'This function is for bivariate data only'
end

X = [ X; 1:N ];
X = sortrows(X')';
original_indices = X(3,:);
X = X([1, 2], :);

mcd_indices = mcd_ext_mex(X, n);
mcd_indices = sort(original_indices(mcd_indices));
```

Contents of `mcd_ext_mex.c`:

```
/*
 * Implementation of Pesch's MCD-EXT algorithm.
 *
 * Geoffrey Robson
 */

#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include "mex.h"

#define Answer(j) (answer[j - 1])
#define Depth(d) (depth[d - 1])
#define UPPER 1
#define LOWER -1
#define NEITHER 0
#define DONE 2
#define ERROR 1
#define OK 0
#define ADD(new, start, traverse) \
    if (start == NULL) { start = new; } \
```

```
    else { traverse->next = new; } \
    traverse = new;
#define SUBSET_ADD(new) subset_traverse->next = new; subset_traverse = new;
#define ON_RIGHT(origin, dest, test) \
    (((test->x - origin->x)*(dest->y - origin->y) \
    > (test->y - origin->y)*(dest->x - origin->x))? 1: 0)

struct point {
    int i;
    double x, y;
    struct point *next;
};

struct point *subset_start, *subset_traverse;

struct info {
    struct point *top, *upper, *middle, *lower, *bottom, *Iu, *Eu, *Il,
        *El, *epsilon;
    int status,
        number_I;
} *depth;

double *answer, *complete, *X;
int N, n;
int d;

/* construction routines */
void add_first_point(int i_1);
int add_second_point(int i_2);
int add_point(void);

/* memory management routines */
struct point* new_point(int index);
void free_all(void);
void free_upper(void);
void free_lower(void);

#ifdef UPDATE_E
void update_E(struct point *aborted_epsilon);
#endif
double calculate_det(void);
int get_epsilon(void);

void mcd_ext(void)
{
    double new_det, min_det = HUGE_VAL;
    int result, j, i_1, i_2;

    depth = (struct info *) malloc(n*sizeof(struct info));

    for (i_1 = 1; i_1 <= N - n + 1; i_1++) {
        add_first_point(i_1);
        for (i_2 = i_1 + 1; i_2 <= N - 1; i_2++) {
            if (add_second_point(i_2) == ERROR) {
```

```
                /* abort criteria already reached */
                free_all();
                continue;
            }
            do {
                while ((result = add_point()) == OK);
                if (result == DONE)
                {
                    /* convex complete subset found */
                    (*complete)++;
                    new_det = calculate_det();
                    if (new_det < min_det) {
                        min_det = new_det;
                        subset_traverse = subset_start;
                        j = 1;
                        while (subset_traverse != NULL) {
                            Answer(j++) = (double) subset_traverse->i;
                            subset_traverse = subset_traverse->next;
                        }
                    }
                    free_all();
                }
            } while (get_epsilon() == OK);
        } /* i_2 */
        free(Depth(1).middle);
    } /* i_1 */
    free(depth);
}

/*
 * Gets the next extreme point and backtracks through depths if necessary.
 */
int get_epsilon(void)
{
    while ((Depth(d).epsilon = Depth(d).epsilon->next) == NULL)
    {
        if (Depth(d).status == UPPER) {
            if ((Depth(d).epsilon = Depth(d).El) != NULL) {
                Depth(d).status = LOWER;
                break;
            }
        }
        d--;
        free_all();
        if (d == 1)
            return DONE;
    }
    return OK;
}

struct point* new_point(int index)
{
    struct point* new;
    double *xptr;
```

```
    if ((new = (struct point *) malloc(sizeof(struct point))) == NULL) {
        fprintf(stderr, "malloc() error!\n");
        exit(1);
    }

    xptr = X + (index - 1)*2;
    new->x = *(xptr++);
    new->y = *xptr;
    new->i = index;
    new->next = NULL;

    return new;
}

void add_first_point(int i_1)
{
    struct point *first = new_point(i_1);

    Depth(1).Iu = NULL;
    Depth(1).Eu = NULL;
    Depth(1).Il = NULL;
    Depth(1).El = NULL;

    Depth(1).middle = first;
    subset_start = first;
    subset_traverse = first;

    d = 1;
}

int add_second_point(int i_2)
{
    struct point
        *traverse_Il = NULL,
        *traverse_Iu = NULL,
        *traverse_El = NULL,
        *traverse_Eu = NULL;
    struct point *second = new_point(i_2);
    struct point *search;
    int number_upper = 0, number_lower = 0, j;

    /* yes, the following initialisations are correct. */
    Depth(2).top = second;
    Depth(2).upper = Depth(1).middle;
    Depth(2).middle = second;
    Depth(2).lower = Depth(1).middle;
    Depth(2).bottom = second;

    Depth(2).Iu = NULL;
    Depth(2).Eu = NULL;
    Depth(2).Il = NULL;
    Depth(2).El = NULL;
```

```
    /* Initialise the future internal regions */
    for (j = Depth(1).middle->i + 1; j < second->i; j++)
    {
        search = new_point(j);
        if (ON_RIGHT(Depth(1).middle, second, search)) {
            ADD(search, Depth(2).Il, traverse_Il);
            number_lower++;
        } else {
            ADD(search, Depth(2).Iu, traverse_Iu);
            number_upper++;
        }
    }

    /* Now initialise the future extreme regions */
    for (j = second->i + 1; j <= N; j++)
    {
        search = new_point(j);
        if (ON_RIGHT(Depth(1).middle, second, search)) {
            ADD(search, Depth(2).El, traverse_El);
            number_lower++;
        } else {
            ADD(search, Depth(2).Eu, traverse_Eu);
            number_upper++;
        }
    }

    SUBSET_ADD(second);
    Depth(2).number_I = 0;

    if (number_upper + 2 < n) {
        // upper region useless
        free_upper();
    }
    if (number_lower + 2 < n) {
        // lower region useless
        free_lower();
    }

    if (Depth(2).Eu != NULL) {
        Depth(2).epsilon = Depth(2).Eu;
        Depth(2).status = UPPER;
    }
    else if (Depth(2).El != NULL) {
        Depth(2).epsilon = Depth(2).El;
        Depth(2).status = LOWER;
    } else {
        return ERROR; /* no future extreme points - terminate */
    }

    d = 2;
    return OK;
}

int add_point(void)
```

```
{
    struct point
        *traverse_Il = NULL,
        *traverse_Iu = NULL,
        *traverse_El = NULL,
        *traverse_Eu = NULL;
    struct point *search, *copy;
    int number_upper = 0, number_lower = 0;

    Depth(d + 1).number_I = Depth(d).number_I;
    Depth(d + 1).Iu = NULL;
    Depth(d + 1).Eu = NULL;
    Depth(d + 1).Il = NULL;
    Depth(d + 1).El = NULL;

    /*
     * Process range A. Separate code for UPPER and LOWER makes for
     * simpler and more readable code.
     */
    if (Depth(d).status == UPPER)
    {
        Depth(d + 1).top = Depth(d).top;
        Depth(d + 1).upper = Depth(d).upper;
        Depth(d + 1).lower = Depth(d).middle;
        Depth(d + 1).bottom = Depth(d).lower;

        /* process the Iu of depth d if UPPER */
        search = Depth(d).Iu;
        while (search != NULL)
        {
            copy = new_point(search->i);
            if (ON_RIGHT(Depth(d).upper, Depth(d).epsilon, copy)) {
                if (++Depth(d + 1).number_I + d == n) {
                    free_all();
                    free(copy);
#ifdef UPDATE_E
                    update_E(Depth(d).epsilon);
#endif
                    return ERROR;
                }
                SUBSET_ADD(copy);
            } else {
                ADD(copy, Depth(d + 1).Iu, traverse_Iu);
                number_upper++;
            }
            search = search->next;
        }
        search = Depth(d).Eu;
    }
    else
    {
        Depth(d + 1).top = Depth(d).upper;
        Depth(d + 1).upper = Depth(d).middle;
        Depth(d + 1).lower = Depth(d).lower;
```

```
        Depth(d + 1).bottom = Depth(d).bottom;

        /* process the Il of depth d if LOWER */
        search = Depth(d).Il;
        while (search != NULL)
        {
            copy = new_point(search->i);/* it must be copied somewhere */
            if (ON_RIGHT(Depth(d).epsilon, Depth(d).lower, copy)) {
                if (++Depth(d + 1).number_I + d == n) {
                    free_all();
                    free(copy);
#ifdef UPDATE_E
                    update_E(Depth(d).epsilon);
#endif
                    return ERROR;
                }
                SUBSET_ADD(copy);
            } else {
                ADD(copy, Depth(d + 1).Il, traverse_Il);
                number_lower++;
            }
            search = search->next;
        }
        search = Depth(d).El;
    }

    /*
     * For ranges B and C, we now use the same code for UPPER and LOWER.
     *
     * Start with range B. ``search'' is already intialised appropriately.
     */
    while (search != NULL)
    {
        if (search->i == Depth(d).epsilon->i) {
            search = search->next; /* move pointer past ``Depth(d).epsilon'' */
            break;
        }
        copy = new_point(search->i);/* it must be copied somewhere */
        if (ON_RIGHT(Depth(d).epsilon, Depth(d + 1).upper, copy)) {
            ADD(copy, Depth(d + 1).Iu, traverse_Iu);
            number_upper++;
        }
        else if (ON_RIGHT(Depth(d + 1).lower, Depth(d).epsilon, search)) {
            ADD(copy, Depth(d + 1).Il, traverse_Il);
            number_lower++;
        }
        else {
            if (++Depth(d + 1).number_I + d == n) {
                free_all();
                free(copy);
#ifdef UPDATE_E
                update_E(Depth(d).epsilon);
#endif
                return ERROR;
```

```
            }
            SUBSET_ADD(copy);
        }
        search = search->next;
    }

    copy = new_point(Depth(d).epsilon->i);
    SUBSET_ADD(copy);

    if (Depth(d + 1).number_I + d + 1 == n) {
        /* convex complete subset of size n found! */
#ifdef UPDATE_E
        update_E(Depth(d).epsilon);
#endif
        return DONE;
    }

    /* Now process range C. */
    while (search != NULL)
    {
        if (ON_RIGHT(Depth(d + 1).upper, Depth(d).epsilon, search)) {
            number_lower++;
            copy = new_point(search->i);
            ADD(copy, Depth(d + 1).El, traverse_El);
        }
        else if (ON_RIGHT(Depth(d).epsilon, Depth(d + 1).lower, search)) {
            number_upper++;
            copy = new_point(search->i);
            ADD(copy, Depth(d + 1).Eu, traverse_Eu);
        }
        search = search->next;
    }

    if (number_upper + Depth(d + 1).number_I + d + 1 < n) {
        free_upper();
    }
    if (number_lower + Depth(d + 1).number_I + d + 1 < n) {
        free_lower();
    }

    if (Depth(d + 1).Eu != NULL) {
        Depth(d + 1).epsilon = Depth(d + 1).Eu;
        Depth(d + 1).status = UPPER;
    }
    else {
        Depth(d + 1).epsilon = Depth(d + 1).El;
        Depth(d + 1).status = LOWER;
    }

    if (Depth(d + 1).epsilon == NULL) {
        free_all();
        return ERROR; /* both Eu and El are empty */
    }
```

```
    Depth(d + 1).middle = subset_traverse;
    d++;
    return OK;
}

/*
 * Remove unusable extreme region
 */
#ifdef UPDATE_E
void update_E(struct point *aborted_epsilon)
{
    struct point *search = aborted_epsilon->next,
        *previous = aborted_epsilon,
        *upper_point,
        *lower_point;

    if (Depth(d).status == UPPER) {
        upper_point = Depth(d).upper;
        lower_point = Depth(d).middle;
    } else {
        upper_point = Depth(d).middle;
        lower_point = Depth(d).lower;
    }

    while (search != NULL)
    {
        if (ON_RIGHT(lower_point, aborted_epsilon, search) &&
            ON_RIGHT(aborted_epsilon, upper_point, search))
        {
            previous->next = search->next;
            free(search);
            search = previous->next;
        } else
        {
            previous = search;
            search = search->next;
        }
    }
}
#endif

/*
 * Free all the info related to depth (d + 1). Does this by freeing
 * the internal points and then calling free_upper() and free_lower()
 */
void free_all(void)
{
    struct point *delete;

    subset_traverse = Depth(d).middle->next;

    while (subset_traverse != NULL) {
        delete = subset_traverse;
        subset_traverse = delete->next;
```

```
        free(delete);
    }

    subset_traverse = Depth(d).middle;
    subset_traverse->next = NULL;

    free_upper();
    free_lower();
}

/*
 * Free the future upper internal and extreme points
 */
void free_upper(void)
{
    struct point *traverse_Iu, *traverse_Eu;
    struct point *delete;

    traverse_Iu = Depth(d + 1).Iu;
    traverse_Eu = Depth(d + 1).Eu;

    while (traverse_Iu != NULL) {
        delete = traverse_Iu;
        traverse_Iu = delete->next;
        free(delete);
    }
    while (traverse_Eu != NULL) {
        delete = traverse_Eu;
        traverse_Eu = delete->next;
        free(delete);
    }

    Depth(d + 1).Eu = NULL;
    Depth(d + 1).Iu = NULL;
}

/*
 * Free the future lower internal and extreme points
 */
void free_lower(void)
{
    struct point *traverse_Il, *traverse_El;
    struct point *delete;

    traverse_Il = Depth(d + 1).Il;
    traverse_El = Depth(d + 1).El;

    while (traverse_Il != NULL) {
        delete = traverse_Il;
        traverse_Il = delete->next;
        free(delete);
    }
    while (traverse_El != NULL) {
        delete = traverse_El;
```

```
        traverse_El = delete->next;
        free(delete);
    }

    Depth(d + 1).Il = NULL;
    Depth(d + 1).El = NULL;
}

/*
 * Calculate the determinant of the covariance matrix of internal points
 */
double calculate_det(void)
{
    double sumx2 = 0,
        sumxy = 0,
        sumy2 = 0,
        sumx = 0,
        sumy = 0;
    double register temp;

    subset_traverse = subset_start;
    while (subset_traverse != NULL)
    {
        sumx2 += subset_traverse->x*subset_traverse->x;
        sumxy += subset_traverse->x*subset_traverse->y;
        sumy2 += subset_traverse->y*subset_traverse->y;
        sumx += subset_traverse->x;
        sumy += subset_traverse->y;
        subset_traverse = subset_traverse->next;
    }
    temp = sumxy - sumy*sumx/n;
    return (sumx2 - sumx*sumx/n)*(sumy2 - sumy*sumy/n) - temp*temp;
}

/*
 * Gateway routine to MATLAB
 */
void mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[])
{
    N = mxGetN(prhs[0]);
    X = mxGetPr(prhs[0]);
    n = mxGetScalar(prhs[1]);

    plhs[0] = mxCreateDoubleMatrix(1, n, mxREAL);
    answer = mxGetPr(plhs[0]);

    /* Two arguments to return, the indices and the #convex complete subsets */
    if (nlhs == 2) {
        plhs[1] = mxCreateDoubleMatrix(1, 1, mxREAL);
        complete = mxGetPr(plhs[1]);
        *complete = 0;
    } else {
        /* simplify the rest of the code by giving the pointer memory */
        complete = malloc(sizeof(double));
```

```
    }

    /* call the actual routine */
    mcd_ext();
}
```

B.5 MCD-BF

Contents of `mcd_bf.m`:

```
%
% The Bernholt-Fischer algorithm for finding the MCD subset.
% The simplest usage is:
%
% mcd_indices = mcd_bf(X, n);
%
% where X is the 2 by N data matrix, n is the subset size, and mcd_indices
% is an array containing the sorted indices of the MCD subset.
%
% Geoffrey Robson 20 March 2003
%

function mcd_indices = mcd_bf(X, n)

[ p, N ] = size(X);

if p ~= 2
error 'This function is for bivariate data only'
end

Z = [ X; X(1,:).*X(2,:); X.*X ];

mcd_indices = mcd_bf_mex(Z, n);

mcd_indices = sort(mcd_indices);
```

Contents of `mcd_bf_mex.c`:

```
/*
 * The polynomial time MCD algorithm of Bernholt and Fischer, with
 * improvements by yours truly.
 *
 * Geoffrey Robson 2003
 */

#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <string.h>
#include "mex.h"
```

```
#define TRUE 1
#define FALSE 0

int N, n;
int f;
int i[5], r[5];
int *selected;

double *Z_start, *answer;
double coef[5];
double a[5][5][5];
double sums_rest[5][5];
double sums_main[5];

/* macros to allow for indices starting at 1 */
#define I(l) (i[l - 1])
#define R(l) (r[l - 1])
#define Selected(j) (selected[j - 1])
#define Sums_main(j) (sums_main[j - 1])
#define A(l,i,j) (a[l - 1][i - 1][j - 2]) /* column one only has 1's and 0's */
#define Sums_rest(i,j) (sums_rest[i - 1][j - 1])
#define Z(i,j) (*(Z_start + (i - 1)*5 + j - 1))
#define Answer(j) (answer[j - 1])

int next_subset(const int set_size, const int subset_size, int *counter);
double calculate_det(int l, const int tau);
int find_selected(int l);
void sums_selected(int tau);

#ifdef UNIQUE

#define NONE 0
#define ABOVE 1
#define BELOW 2

struct tree_node {
    int *start_index;
    struct tree_node *parent;
    int status;
    struct tree_node *above;
    struct tree_node *below;
};
struct tree_node *root;

int *compare;
double *total, *unique;

int trivial(const int *a, const int *b);
void test_subset(void);
void add_subset(struct tree_node **new, struct tree_node *parent, int status);
void free_tree(void);
#endif /* UNIQUE */
```

```
void mcd_bf(void)
{
    int l_main,  /* level of (N choose f) subset tree */
        l_rest,  /* level of (f choose n - tau) subset tree */
        tau, j;
    double min_det = HUGE_VAL, new_det;

    selected = (int *) malloc(n*sizeof(int));
#ifdef UNIQUE
    root = NULL;
    *total = 0;
    *unique = 0;
    compare = (int *) malloc((n + 1)*sizeof(int));
    compare[n] = 1; /* terminated with a one to force != unique_scan[n] */
    puts("Warning: compiled with -DUNIQUE option");
#endif /* UNIQUE */

    for (f = 5; f >= 4; f--)
    {
        for (j = 1; j <= f; j++)
            I(j) = j;
        l_main = 1;

        do {
            if ((tau = find_selected(l_main)) != n)
            {
                sums_selected(tau);

                for (j = 1; j <= n - tau; j++)
                    R(j) = j;
                l_rest = 1;

                do {
#ifdef UNIQUE
                    (*total)++;
                    for (j = 1; j <= tau; j++)
                        *(compare++) = Selected(j);
                    for (j = tau + 1; j <= n; j++)
                        *(compare++) = I(R(j - tau));
                    compare -= n;
                    qsort((void *) compare, n, sizeof(int),
                        (int (*)(const void *, const void *)) trivial);
                    test_subset();
#endif /* UNIQUE */
                    new_det = calculate_det(l_rest, tau);

                    if (new_det < min_det)
                    {
                        min_det = new_det;
                        for (j = 1; j <= tau; j++)
                            Answer(j) = (double) Selected(j);
                        for (j = tau + 1; j <= n; j++)
                            Answer(j) = (double) I(R(j - tau));
```

```
                    }
                } while ((l_rest = next_subset(f, n - tau, &r[0])));
            }
        } while ((l_main = next_subset(N, f, &i[0])));
    }
    free(selected);
#ifdef UNIQUE
    free(compare);
    free_tree();
#endif /* UNIQUE */
}

/*
 * Create a new subset
 */
int next_subset(const int set_size, const int subset_size, int *counter)
{
    int j, l = subset_size;
#define Counter(l) (counter[l - 1])

    while (++Counter(l) > set_size - subset_size + l) {
        if (l-- == 1)
            return 0;
    }
    for (j = l + 1; j <= subset_size; j++)
        Counter(j) = Counter(j - 1) + 1;
    return l;
}

/*
 * Find the sums of the tau points.
 */
void sums_selected(int tau)
{
    double *Z_ptr, *sums_ptr = &sums_main[0];
    int j;

    *(sums_ptr++) = 0;
    *(sums_ptr++) = 0;
    *(sums_ptr++) = 0;
    *(sums_ptr++) = 0;
    *sums_ptr = 0;
    sums_ptr -= 4;
    for (j = 1; j <= tau; j++) {
        Z_ptr = Z_start + (Selected(j) - 1)*5;
        *(sums_ptr++) += *(Z_ptr++);   /*  + x    */
        *(sums_ptr++) += *(Z_ptr++);   /*  + y    */
        *(sums_ptr++) += *(Z_ptr++);   /*  + xy   */
        *(sums_ptr++) += *(Z_ptr++);   /*  + x^2  */
        *sums_ptr += *(Z_ptr++);       /*  + y^2  */
        sums_ptr -= 4;
    }
}
```

```
/*
 * Calculate the determinant by adding the (n - tau) points to the sums
 * produced by the tau points.
 */
double calculate_det(int l, const int tau)
{
    double register temp;
    int k;
    int m = n - tau;

    if (l == 1) {
        for (k = 1; k <= 5; k++)
            Sums_rest(1, k) = Z(I(R(1)), k);
        l++;
    }

    while (l <= m)
    {
        for (k = 1; k <= 5; k++)
            Sums_rest(l, k) = Sums_rest(l - 1, k) + Z(I(R(l)), k);
        l++;
    }

    for (k = 1; k <= 5; k++)
        Sums_rest(m, k) += Sums_main(k);

    /* temp = sum(xy) - sum(x)*sum(y)/n; */
    temp = Sums_rest(m, 3) - Sums_rest(m, 2)*Sums_rest(m, 1)/n;

    /* return (sum(x^2) - sum(x)*sum(x)/n)
            *(sum(y^2) - sum(y)*sum(y)/n) - temp^2; */
    return (Sums_rest(m, 4) - Sums_rest(m, 1)*Sums_rest(m, 1)/n)
        *(Sums_rest(m, 5) - Sums_rest(m, 2)*Sums_rest(m, 2)/n) - temp*temp;
}

/*
 * Add all the new rows to the matrix A, and then reduce them to form
 * an identity matrix. If C' > 0, then calculate and return the number
 * the number of points selected.
 */
int find_selected(int l)
{
    int j, k, tau = 0;
    double register temp;
    double *coef_ptr = &coef[0];
    double *Z_ptr = Z_start;

    if (l == 1)
    {
        /* Copy the first row from Z into A */
        for (k = 1; k <= 3; k++)
            A(1, 1, k + 1) = Z(I(1), k);    /* x, y, xy */
        if (f == 5)
            A(1, 1, 5) = Z(I(1), 5);        /* y^2 */
```

```
        A(1, 1, f + 1) = -Z(I(1), 4);       /* -x^2 */
        l++;
    }

    while (1)
    {
        /* Copy the new row from Z into A */
        for (k = 1; k <= 3; k++)
            A(l, l, k + 1) = Z(I(l), k);    /* x, y, xy */
        if (f == 5)
            A(l, l, 5) = Z(I(l), 5);        /* y^2 */
        A(l, l, f + 1) = -Z(I(l), 4);       /* -x^2 */

        /* Transform A to upper triangular by reducing the new row
         * (The first column contains 1, so treat specially) */
        for (k = l; k <= f + 1; k++)
            A(l, l, k) -= A(l - 1, 1, k);
        for (j = 2; j < l; j++) {
            temp = A(l, l, j);
            for (k = l; k <= f + 1; k++)
                A(l, l, k) -= temp*A(l - 1, j, k);
        }
        /* 1 on diagonal */
        temp = A(l, l, l);
        for (k = l + 1; k <= f + 1; k++)
            A(l, l, k) /= temp;

        if (l == f)
        {
            /* is f == 5 and C' < 0 ? */
            if ((f == 5) && (A(5, 5, 6) < 0))
                return n;

            /* find the coefficients */
            for (j = 1; j < f; j++) {
                *(coef_ptr++) = A(f - 1, j, f + 1) -
                    A(f - 1, j, f)*A(f, f, f + 1);
            }
            *(coef_ptr++) = A(f, f, f + 1);
            coef_ptr -= f;
            break;
        }
        else
        {
            /* We now use this fully reduced row to reduce the previous
             * rows. i.e. Gauss-Jordan form */
            for (j = 1; j < l; j++) {
                temp = A(l - 1, j, l);
                for (k = l + 1; k <= f + 1; k++)
                    A(l, j, k) = A(l - 1, j, k) - temp*A(l, l, k);
            }
        }
        l++;
    }
```

```
    /* calculate how many are selected by the quadratic */
    l = 1;
    for (j = 1; j <= N; j++)
    {
        /* we avoid evaluating the quadratic for those points on
         * the boundary */
        if (j == I(l)) {
            if (n - f - tau > N - j)
                return n;
            l++;
            Z_ptr += 5;
        }
        else
        {
            /*    Z = [ x, y, xy, x^2, y^2 ] */
            /* coef = [ F', D', E', B', C' ] */
            temp = *(coef_ptr++);               /*  = F'      */
            temp += *(Z_ptr++)*(*(coef_ptr++)); /*  + D'*x    */
            temp += *(Z_ptr++)*(*(coef_ptr++)); /*  + E'*y    */
            temp += *(Z_ptr++)*(*(coef_ptr++)); /*  + B'*xy   */
            temp += *(Z_ptr++);                 /*  + x^2     */
            if (f == 5)
                temp += *(Z_ptr)*(*(coef_ptr)); /*  + C'*y^2  */
            Z_ptr++;
            coef_ptr -= 4;

            if (temp < 0) {
                if (++tau == n)
                    return n;
                Selected(tau) = j;
            }
            else if (n - f - tau > N - j)
                return n;
        }
    }
    return tau;
}

#ifdef UNIQUE
int trivial(const int *a, const int *b)
{
    return (*a - *b);
}

void test_subset(void)
{
    const int *unique_scan, *compare_scan;
    struct tree_node *temp;

    /* ``unique_scan'' is terminated with a zero,
     * and ``compare'' with a 1 */

    if (root == NULL) {
```

```
        add_subset(&root, NULL, NONE);
        return;
    }
    else
        temp = root;

    while (1) {
        compare_scan = compare;
        unique_scan = temp->start_index;
        while (*unique_scan == *compare_scan) {
            compare_scan++;
            unique_scan++;
        }

        if (*unique_scan == 0) {
            return; /* compare already exists in tree */
        }

        if (*compare_scan > *unique_scan) {
            if (temp->above == NULL) {
                add_subset(&temp->above, temp, ABOVE);
                return;
            }
            temp = temp->above;
        } else {
            if (temp->below == NULL) {
                add_subset(&temp->below, temp, BELOW);
                return;
            }
            temp = temp->below;
        }
    }
}

void add_subset(struct tree_node **new, struct tree_node *parent, int status)
{
    *new = (struct tree_node *) malloc(sizeof(struct tree_node));
    (*new)->start_index = malloc((n + 1)*sizeof(int));
    memcpy((*new)->start_index, compare, n*sizeof(int));
    *((*new)->start_index + n) = 0;
    (*new)->above = NULL;
    (*new)->below = NULL;
    (*new)->parent = parent;
    (*new)->status = status;
    (*unique)++;
}

void free_tree(void)
{
    struct tree_node *parent, *temp = root;

    while (temp != NULL) {
        if (temp->above != NULL) {
            temp = temp->above;
```

```
        } else if (temp->below != NULL) {
            temp = temp->below;
        } else {
            parent = temp->parent;
            if (temp->status == ABOVE)
                parent->above = NULL;
            else if (temp->status == BELOW)
                parent->below = NULL;
            free(temp->start_index);
            free(temp);
            temp = parent;
        }
    }
}
#endif /* UNIQUE */

/*
 * Gateway routine to MATLAB
 */
void mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[])
{
    N = mxGetN(prhs[0]);
    Z_start = mxGetPr(prhs[0]);
    n = mxGetScalar(prhs[1]);

    plhs[0] = mxCreateDoubleMatrix(1, n, mxREAL);
    answer = mxGetPr(plhs[0]);

#ifdef UNIQUE
    if (nlhs == 3) {
        plhs[1] = mxCreateDoubleMatrix(1, 1, mxREAL);
        total = mxGetPr(plhs[1]);
        *total = 0;
        plhs[2] = mxCreateDoubleMatrix(1, 1, mxREAL);
        unique = mxGetPr(plhs[2]);
        *unique = 0;
    } else {
    /* simplify the rest of the code by giving the pointers something
     * to point at */
        unique = malloc(sizeof(double));
        total = malloc(sizeof(double));
    }
#endif /* UNIQUE */

    /* call the actual routine */
    mcd_bf();
}
```