Swiss Federal Institute of Technology Zurich

Seminar for Statistics

Department of Mathematics

Bachelor Thesis Winter 2019

Nicolas Trutmann

Comparison of EM-algorithm and MLE using Cholesky decomposition

Submission Date: placeholder

Advisor: placeholder

Abstract

The intent of this work is to compare The EM algorithm to a MLE approach in the case of multivariate normal mixture models. using the Cholesky decomposition. The EM algorithm is widely used in statistics and is proven to converge, however in pathological cases convergence slows down considerably.

We compare two implementations of each algorithm with two different initialization strategies by judging their performance in model selection. Model selection is decided by the Bayesian Information Criterion (BIC).

The results are promising. In many cases MLE is equal or better than EM. It is certainly a competitive model selection strategy.

Contents

1 Introduction to normal mixture models **1**
1.1 Definitions . . . 1
1.2 The EM-Algorithm in Sketch . . . 2
1.3 Choice of Notation . . . 3
1.4 Models of Covariance Matrices . . . 4
1.5 Problems of the EM-algorithm . . . 6
1.6 Alternative Option . . . 7

2 The `norMmix` Package **9**
2.1 Introduction to the Package . . . 9
2.1.1 `norMmixMLE` . . . 10
2.2 On The Development of `norMmix` . . . 12
2.3 Demonstration . . . 13

3 Comparing Algorithms **15**
3.1 Time Analysis . . . 17
3.2 Behaviour in `n` . . . 20
3.3 Difficult Mixtures . . . 23
3.4 Nonnormal Mixtures . . . 29

4 Discussion **33**
4.1 Acknowledgements . . . 33

Bibliography **34**

A R Code **37**
A.1 `llnorMmix` . . . 37
A.2 Example Simulation Script . . . 41

B Further Plots **43**
B.1 Behaviour in n . . . 43
B.2 Unused Data . . . 43

List of Figures

1.1 Parameters of `MW.nm9` . . . 7
1.2 True and Estimated density . . . 7
1.3 20 EM steps . . . 7
1.4 200 EM steps . . . 7
1.5 Log-likelihood Plotted against Iteration Count for the Example in 1.5 . . . 8

2.1 Demonstration of the MW Object `MW215`. Correct model: `model="VEE", k=3` 13
2.2 Correct Mixture (left) and Fitted overlayed in orange (right) . . . 14

3.1 Example of Comparison Plot . . . 16
3.2 Log-log Plot of System Time against Parameter Length . . . 19
3.3 The mixture model `MW34`, a three dimensional, two component mixture with one smaller, lesser weighted component inside a smaller one. . . . 20
3.4 BIC values of `MW34` with $n = 2000$ . . . 21
3.5 BIC values of `MW34` with $n = 2000$ . . . 22
3.6 Trimodal mixture `MW215`. Three equally weighted, oriented, and shaped components of different volumes along the diagonal . . . 23
3.7 BIC values of `MW34`, correct: `model="VVI", k=2` . . . 24
3.8 Claw-like mixture . . . 25
3.9 BIC values of claw-like mixture. Best fit: `model="VEE", k=8`, correct: model=”VII”, k=6 . . . 26
3.10 Best Fit over n = model selections. `model="VEE", k=8` Correct model `model="VII", k=6`. Of Note Here are the Spurious Clusters Appearing. . . 27
3.11 Two of the better clusters. They both follow the ’patchwork’ covering strategy, laying patches of components over the dataset . . . 28
3.12 The BIC values of the `SMI.12` data. The blue line representing the `clara` values is covered by the other lines. The last three models are not plotted for all component sizes, as explained in text 3.4 . . . 29
3.13 Truncated Iris . . . 31
3.14 Loss data . . . 32

B.1 BIC values of `MW34` with $n = 2000$ . . . 44
B.2 BIC values of `MW34` with $n = 2000$ . . . 45
B.3 BIC values of `MW51` with $n = 2000$ . . . 46
B.4 BIC values of `MW34` with $n = 2000$ . . . 47

List of Tables

1.1 Table of Parameters of the Covariance Matrices 5
1.2 Full Table of Parameters 6

2.1 Translation Table: Mathematical Notation to R Code 9

Chapter 1

Introduction to normal mixture models

1.1 Definitions

A good and thorough introductory book is the work of McLachlan and Peel (2000) and the reader is encouraged to study it to learn in depth about normal mixtures and clustering. We will here give a short overview of normal mixtures to fix notation and nomenclature. The motivating idea behind mixture models is, that in real world examples a sample might be suspected to arise from more than one population or be more simply modelled by several overlayed distributions. The example of this, that is generally considered to be the first of this kind, is the one by Karl Pearson, who fitted two normal distributions with different means and variances. In his book, Pearson and Henrici (1896)[Section 4.d.; page 266], Pearson analyzed measurements of forehead to body length of crabs sampled from the bay of Naples. His mixture model-based approach suggested, that the crabs were evolving into two new subspecies. This is a historically important example, because it presents statistical evidence of evolution in process. Mixture models have been used since, but research took off after the availability of computing power made computational research possible

While the theory of mixture models holds for a much broader class of distributions, we restrict ourselves here to the case of normal distributions, because this restriction fits more comfortably into the scope of this work and because normal distributions allow for a parsimonious parametrization, that is of interest to study.

This parametrization is the $\boldsymbol{LDL}^\top$ decomposition, which allows a very simple parametrization and a straightforward connection between degrees of freedom and necessarily generated numerical values. This will be explained further in section 1.4.

But before we delve deeper into the topic of this research, we first define the concept of a normal mixture model:

Let $\mu \in \mathbb{R}^p, \quad \Sigma \in \mathbb{R}^{p \times p}$ be symmetric positive definite and $\phi(-; \mu, \Sigma)$ be the normal distribution with mean μ and covariance matrix Σ with density function:

$$\phi(\boldsymbol{x};\boldsymbol{\mu},\boldsymbol{\Sigma}) = \frac{\exp(-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})\boldsymbol{\Sigma}^{-\frac{1}{2}}(\boldsymbol{x}-\boldsymbol{\mu})^{\top})}{\sqrt{(2\pi)^k \det \boldsymbol{\Sigma}}} \tag{1.1.0.1}$$

for $\boldsymbol{x} \in \mathbb{R}^p$. Since we are studying mixture models, we will need several overlapping of normal distributions, of differing means and covariance. Therefore, we choose notation allowing us to refer to the components in shorthand. Let us assume we have $K \in \mathbb{N}$ normal distributions with means and covariance $\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k, \quad k \in \{1,\ldots,K\}$, then we fix:

$$\phi_k(\boldsymbol{x}) := \phi(\boldsymbol{x};\boldsymbol{\mu_k},\boldsymbol{\Sigma_k}) \tag{1.1.0.2}$$

And going forward, we will refer to components by the subscript k.

Definition 1.1.0.1. *Suppose we have a random sample $\boldsymbol{Y}_1,\ldots,\boldsymbol{Y}_n$, where $\boldsymbol{Y}_i$ is a p-dimensional random vector with probability density function $\boldsymbol{Y}_i \sim f(\boldsymbol{y}_i)$ on $\mathbb{R}^p$.*

We assume that the density $f(\boldsymbol{y}_i)$ of $\boldsymbol{Y}_i$ can be written in the form:

$$f(\boldsymbol{y}_i) = \sum_{k=1}^{K} \pi_k \phi_k(\boldsymbol{y}_i) \tag{1.1.0.3}$$

The ϕ_k are normal distributions and are called the mixture components with parameters $\boldsymbol{\mu}_k$ and $\boldsymbol{\Sigma}_k$ as described above (1.1.0.2). The π_k are called the component densities of the mixture and are constrainded by the rules $\pi_k > 0$ and $\sum_k \pi_k = 1$.

For 'large' datasets there are more parsimonious parametrizations, that reduce computation time. These, for example, assume that all components have the same covariance, or have certain restrictions placed on them. We will give a detailed description of the models assumed in this thesis in section 1.4.

1.2 The EM-Algorithm in Sketch

With this definition, we immediately face the problem of how to fit these mixture components to given data. A popular algorithm to solve this problem is the **E**xpectation-**M**aximization algorithm, abbreviated as EM-algorithm.

We give here a sketch of the EM-algorithm in the case of all normal mixture components. This roughly follows the content in McLachlan and Peel (2000). For a more thorough treatment of the matter see chapter 3.

Suppose we have a $p-$dimensional dataset of n samples $x_1,\ldots,x_n$, onto which we would like to fit a K component normal mixture with mixture components $\phi_k,\ k \in 1,\ldots,n$.

For the EM-algorithm further parameters are introduced. These are denoted $\tau_j(\boldsymbol{y}_i)$ and they represent the posterior probabilities that observation i is a member of component j.

The EM-algorithm is a two step, iterative process consisting of an 'e'-step and an 'm'-step. In the e-step the expectation of component membership is updated.

$$\tau_j(\boldsymbol{y}_i;\Psi) = \phi_j(\boldsymbol{y}_i) / \sum_{k=1}^{K} \phi_k(\boldsymbol{y}_i) \tag{1.2.0.1}$$

and in the m-step given the component membership information we update the component means and covariances by weighted versions of the usual estimators.

$$\boldsymbol{\mu}_j = \sum_{i=1}^{n} \tau_j(\boldsymbol{y}_i)\boldsymbol{y}_i / \sum_{j=1}^{n} \tau_j(\boldsymbol{y}_i) \tag{1.2.0.2}$$

$$\boldsymbol{\Sigma}_j = \sum_{i=1}^{n} \tau_j(\boldsymbol{y}_i)(\boldsymbol{y}_i - \boldsymbol{\mu}_j)(\boldsymbol{y}_i - \boldsymbol{\mu}_j)^{\top} / \sum_{i=1}^{n} \tau_j(\boldsymbol{y}_i) \tag{1.2.0.3}$$

There remains to be stated how to start the algorithm. Since both steps of the algorithm depend on data from the other, the EM-algorithm needs some form of initialization step. Most popular implementations use some form of pre clustering and use the EM-algorithm as subsequent tools to fit the data. The R-package `mclust` for example uses hierarchical agglomerative clustering Scrucca, Fop, Murphy, and Raftery (2016).

1.3 Choice of Notation

The classification of models in this paper relies heavily on the work of Celeux and Govaert (1995), however, out of necessity for clarity, we break with their notation. So as to not confuse the reader we describe here in depth the differences in notation between Celeux and Govaert (1995) and ours.

The basis of classification in Celeux and Govaert (1995) is the decomposition of a symmetric matrix into an orthogonal and a diagonal component. A symmetric positive definite matrix Σ can be decomposed as follows

$$\Sigma = \lambda \boldsymbol{D}\boldsymbol{A}\boldsymbol{D}^{\top} \tag{1.3.0.1}$$

with $\boldsymbol{D}$ an orthogonal matrix and $\boldsymbol{A}$ a diagonal matrix and $\lambda = \sqrt[p]{det(\Sigma)}$ the $p - th$ root of the determinant of Σ.

This decomposition has an appealing geometric interpretation, with $\boldsymbol{D}$ as the *orientation* of the distribution, $\boldsymbol{A}$ the *shape*, and λ the *volume*. The problem of notation comes from standard conventions in linear algebra, where the letters A and D are usually occupied by arbitrary and diagonal matrices respectively. Furthermore, we intend to apply a variant of the Cholesky decomposition to Σ, the $\alpha\boldsymbol{L}\boldsymbol{D}\boldsymbol{L}^{\top}$ decomposition. This obviously raises some conflicts in notation.

Therefore we, from here on, when referring to the decomposition as described by Celeux and Govaert (1995), will use the following modification of notation:

$$\boldsymbol{D} \longmapsto \boldsymbol{Q} \tag{1.3.0.2}$$
$$\boldsymbol{A} \longmapsto \boldsymbol{\Lambda} \tag{1.3.0.3}$$
$$\lambda \longmapsto \alpha \tag{1.3.0.4}$$
$$\Sigma = \lambda \boldsymbol{D}\boldsymbol{A}\boldsymbol{D}^{\top} = \alpha \boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^{\top} \tag{1.3.0.5}$$

These were chosen according to general conventions of linear algebra. $\boldsymbol{Q}$ is usually chosen for orthonormal matrices; $\boldsymbol{\Lambda}$ is often a choice for diagonal matrices of eigenvectors and α was somewhat arbitrarily chosen.

1.4 Models of Covariance Matrices

As mentioned above, there are ways to constrain the covariance matrices for computational reasons. There are istances where the resulting loss of information is seen as acceptable, for example if, through consideration of the data, that a simplified model is acceptable. Another is if the sheer size of the data makes application of full generality impossible.

- We restrict the complexity of the decomposition components
- We restrict the variability of the mixture components

Let us look at the first case. We take the decomposition of a covariance matrix as $\Sigma = \alpha \boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^\top$. Of these, we can simplify the structure of $\boldsymbol{Q}$ and $\boldsymbol{\Lambda}$, by replacing them with the identity. If we set $\boldsymbol{Q} = \mathrm{Id}$, we lose the freedom of orientation and if we set $\boldsymbol{\Lambda} = \mathrm{Id}$ we restrict ourselves to spherical distributions.

of course, we cannot restrict $\boldsymbol{\lambda}$ while letting $\boldsymbol{q}$ free, since

$$\boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^\top = \boldsymbol{Q}\mathrm{Id}\boldsymbol{Q}^\top = \mathrm{Id} \tag{1.4.0.1}$$

The second restriction simply means we hold the decomposition fixed throughout all covariance matrices. There is however an issue with the cholesky decomposition. For 10 out of 14 cases as defined by Celeux and Govaert (1995), there exists a canonical translation of decompositions. The 6 diagonal cases need no translation; the eigen and cholesky decomposition are equal to identity. For the non-diagonal cases note that for a given sym. pos. def. matrix Σ we have decompositions:

$$\Sigma = \alpha \boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^\top \quad \Sigma = \alpha \boldsymbol{L}\boldsymbol{D}\boldsymbol{L}^\top \tag{1.4.0.2}$$

Since in both cases the enclosing matrices $\boldsymbol{Q}$ and $\boldsymbol{L}$ have determinant 1 the determinant of Σ falls entirely on α. therefore α, in these particular decompositions, is equal for both. Celeux and Govaert (1995) vary σ by either varying or holding fixed the volume (α/α_k), shape $(\boldsymbol{\Lambda}/\boldsymbol{\Lambda_k})$ and orientation $(\boldsymbol{Q}/\boldsymbol{Q}_k)$. These 3 times 2 cases would yield the 8 out of 14 cases of non-diagonal cases. However there is no canonical transform for either variable orientation and fixed shape or fixed orientation and variable shape. The reason for this is that in the $\boldsymbol{L}\boldsymbol{D}\boldsymbol{L}^\top$ decomposition the lower diagonal matrix $\boldsymbol{L}$ holds some of the shape of the matrix, which in the eigendecomposition is in the $\boldsymbol{\Lambda}$ matrix. In fact, $\boldsymbol{L}$ is orthogonal if and only if $\boldsymbol{L} = \mathrm{Id}_{n\times n}$. Therefore we can only decompose matrices where either both or neither shape and orientation vary. See table 1.1.

While we could in theory construct the cases $\boldsymbol{L}\boldsymbol{D}_k\boldsymbol{L}^\top$ and $\boldsymbol{L}_k\boldsymbol{D}\boldsymbol{L}_k^\top$, however they do not correspond to the desired geometric intent behind the differentiation of models and are therefore not included.

Model	$\boldsymbol{\Sigma}_k$ C&G	volume	shape	orientation	parameters	$\boldsymbol{LDL}^\top$	parameters	count
EII	$\alpha \boldsymbol{I}$	equal	equal	-	α	as in C&G		1
VII	$\alpha_k \boldsymbol{I}$	var.	equal	-	α_k			K
EEI	$\alpha \boldsymbol{\Lambda}$	equal	equal	coord. axes	α, λ_i			$1 + (p-1)$
VEI	$\alpha_k \boldsymbol{\Lambda}$	var.	equal	coord. axes	α_k, λ_i			$K + (p-1)$
EVI	$\alpha \boldsymbol{\Lambda}_k$	equal	var.	coord. axes	$\alpha, \lambda_{i,k}$			$1 + K(p-1)$
VVI	$\alpha_k \boldsymbol{\Lambda}_k$	var.	var.	coord. axes	$\alpha_k, \lambda_{i,k}$			$K + K(p-1)$
EEE	$\alpha \boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^\top$	equal	equal	equal	$\alpha, \lambda_i, q_{i,j}$	$\alpha \boldsymbol{LDL}^\top$	$\lambda, d_i, l_{i,j}$	$1 + (p-1) + \frac{p(p-1)}{2}$
EVE	$\alpha \boldsymbol{Q}\boldsymbol{\Lambda}_k\boldsymbol{Q}^\top$	equal	var.	equal	$\alpha, \lambda_{i,k}, q_{i,j}$	doesn't exist		$1 + K(p-1) + \frac{p(p-1)}{2}$
VEE	$\alpha_k \boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^\top$	var.	equal	equal	$\alpha_k, \lambda_i, q_{i,j}$	$\alpha_k \boldsymbol{LDL}^\top$	$\lambda_k, d_i, l_{i,j}$	$K + (p-1) + \frac{p(p-1)}{2}$
VVE	$\alpha_k \boldsymbol{Q}\boldsymbol{\Lambda}_k\boldsymbol{Q}^\top$	var.	var.	equal	$\alpha_k, \lambda_{i,k}, q_{i,j}$			$K + K(p-1) + \frac{p(p-1)}{2}$
EEV	$\alpha \boldsymbol{Q}_k\boldsymbol{\Lambda}\boldsymbol{Q}_k^\top$	equal	equal	var.	$\alpha, \lambda_i, q_{i,j,k}$	don't exist		$1 + (p-1) + K\frac{p(p-1)}{2}$
VEV	$\alpha_k \boldsymbol{Q}_k\boldsymbol{\Lambda}\boldsymbol{Q}_k^\top$	var.	equal	var.	$\alpha_k, \lambda_i, q_{i,j,k}$			$K + (p-1) + K\frac{p(p-1)}{2}$
EVV	$\alpha \boldsymbol{Q}_k\boldsymbol{\Lambda}_k\boldsymbol{Q}_k^\top$	equal	var.	var.	$\alpha, \lambda_i, q_{i,j,k}$	$\alpha \boldsymbol{L}_k\boldsymbol{D}_k\boldsymbol{L}_k^\top$	$\lambda, d_{i,k}, l_{i,j,k}$ $j > i$	$1 + K(p-1) + K\frac{p(p-1)}{2}$
VVV	$\alpha_k \boldsymbol{Q}_k\boldsymbol{\Lambda}_k\boldsymbol{Q}_k^\top$	var.	var.	var.	$\alpha_k, \lambda_i, q_{i,j,k}$	$\alpha_k \boldsymbol{L}_k\boldsymbol{D}_k\boldsymbol{L}_k^\top$	$\lambda_k, d_{i,k}, l_{i,j,k}$ $j > i$	$K + K(p-1) + K\frac{p(p-1)}{2}$

Table 1.1: Table of Parameters of the Covariance Matrices

Σ model	μ, π	Σ	total #{par}	$\mathcal{O}()$
EII	$K-1+pK$	1	$Kp+K$	Kp
VII	$K-1+pK$	K	$Kp+2K-1$	Kp
EEI	$K-1+pK$	$1+(p-1)$	$Kp+p+K-1$	Kp
VEI	$K-1+pK$	$K+(p-1)$	$Kp+p+2K-2$	Kp
EVI	$K-1+pK$	$1+K(p-1)$	$2Kp$	Kp
VVI	$K-1+pK$	$K+K(p-1)$	$2Kp+K-1$	Kp
EEE	$K-1+pK$	$1+(p-1)+\frac{p(p-1)}{2}$	$\frac{(p+2)(p-1)}{2}+Kp+K$	p^2+Kp
VEE	$K-1+pK$	$K+(p-1)+\frac{p(p-1)}{2}$	$\frac{(p+2)(p-1)}{2}+Kp+2K-2$	p^2+Kp
EVV	$K-1+pK$	$1+K(p-1)+K\frac{p(p-1)}{2}$	$K\frac{(p+2)(p-1)}{2}+Kp+K$	Kp^2
VVV	$K-1+pK$	$K+K(p-1)+K\frac{p(p-1)}{2}$	$K\frac{(p+2)(p-1)}{2}+Kp+2K-1$	Kp^2

Table 1.2: Full Table of Parameters

There is an attractive advantage in using the $\boldsymbol{LDL}^\top$ decomposition. Since both the $\boldsymbol{LDL}^\top$ and eigendecomposition derive from the same covariance matrix, the necessary parameters are the same in cardinality. In the case of the $\boldsymbol{Q}$ and $\boldsymbol{L}$ matrices, there need to be $\frac{p(p-1)}{2}$ parameters to be determined to uniquely define these matrices. In the case of the $\boldsymbol{L}$ matrix these are straightforward the entries of the lower diagonal matrix, whereas $\boldsymbol{Q}$ needs a nontrivial amount of work to determine a minimal generating set of parameters, which makes computation of the decomposition as in Celeux and Govaert (1995) a lot more difficult. Therefore the $\boldsymbol{LDL}^\top$ decomposition was chosen for the purpose of this thesis.

1.5 Problems of the EM-algorithm

The EM-algorithm has stalling problems especially close to a local optimum. In their seminal work, Dempster, Laird, and Rubin (1977), have proven that the EM-algorithm converges under mild regularity conditions. However, convergence does not guarantee fast convergence. In fact, a lot of the work, that has gone into the research around the EM-algorithm has been concerned with speeding up convergence, see McLachlan and Peel (2000)[section 2.17]. The concern here is that a slowing in convergence might be mistaken for actual convergence.

This phenomenon is not infrequent and in difficult mixtures quite visible. To illustrate let us look at a particular mixture taken from Marron and Wand (1992) and the `nor1mix` package from CRAN. `nor1mix` is a package designed and developed for educational purposes to teach about univariate normal mixtures. It is also the spiritual predecessor of this thesis' R code.

The mixture is a trimodal mixture of uneven weight, as shown in figure 1.2. While not the most difficult mixture studied by Marron and Wand (1992), it is certainly not trivial either. In the figures below, 1.5 and 1.5, we demonstrate, that even after 200 iterations of the EM-algorithm the convergence is poor. In this instance, the initialization is done using R 's CLARA implementation from the cluster package.

then an illustration of MW examples of pathological cases

We can see, that the EM-algorithm seems to converge to an intermediary solution, where the smaller middle solution is weighted lower, until it manages to correct back and find

```
>    library("nor1mix")
>    MW.nm9 ## Trimodal mixture
'Normal Mixture' object    ``#9 Trimodal''
       mu sigma    w
[1,] -1.2  0.60 0.45
[2,]  1.2  0.60 0.45
[3,]  0.0  0.25 0.10
```

Figure 1.1: Parameters of `MW.nm9`

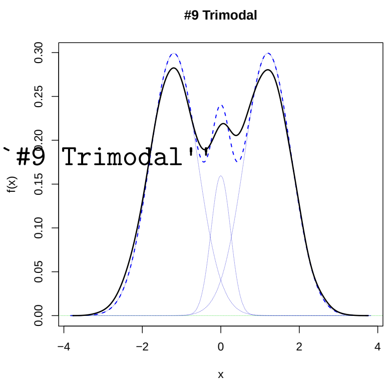


Figure 1.2: True and Estimated density

the correct components.

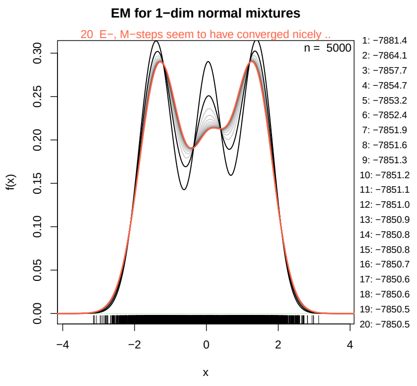


Figure 1.3: 20 EM steps

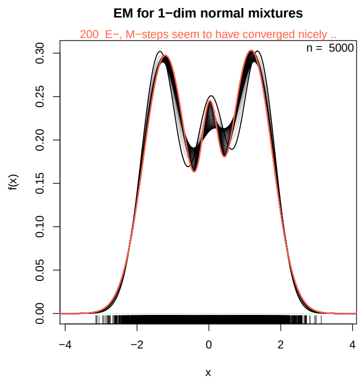


Figure 1.4: 200 EM steps

We see how change in log-likelihood seems to stagnate. However, this does not stay that way. If we let EM run a bit further we see, the log-likelihood hits a flatspot, after which convergence accelerates again.

In fact, it seems that the previous solution is a saddle point in the likelihood function, where EM has chronic problems continuing improvements.

give 2D demonstration.

1.6 Alternative Option

In conclusion, the EM-algorithm has very appealing advantages. However, as we have shown, there are chronic problems in convergence rates. The aim of this thesis is to test if some improvement could be achieved by a different method.

The plan is reasonably straightforward:

i.) Initialize using CLARA.

ii.) Perform one m-step, to transform CLARA's results into the form of a normal mixture.

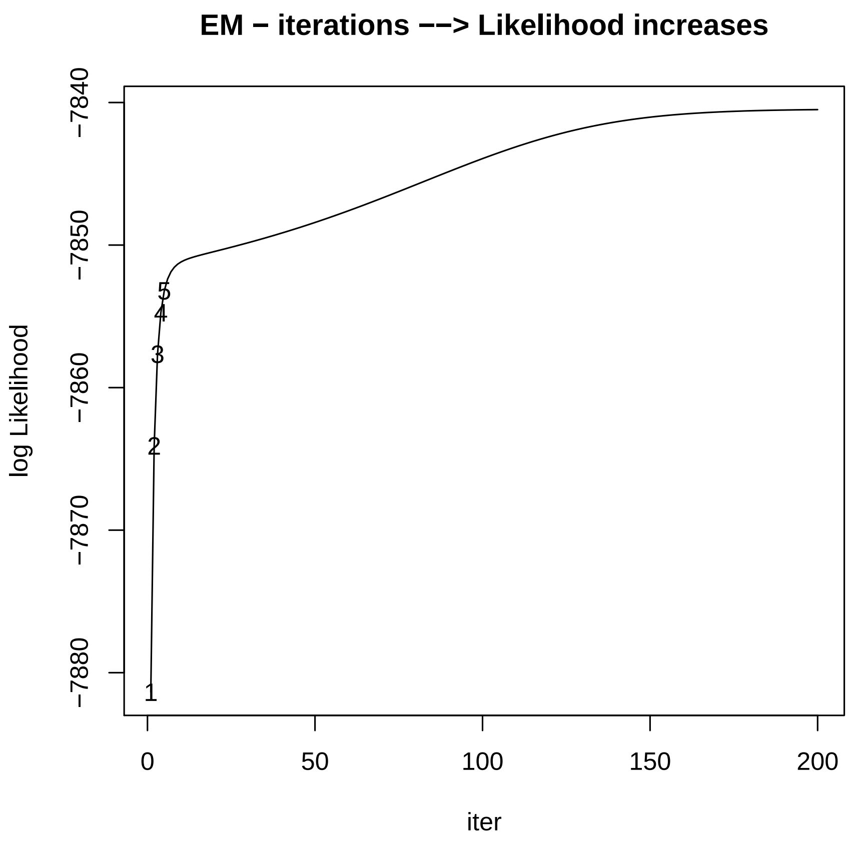


Figure 1.5: Log-likelihood Plotted against Iteration Count for the Example in 1.5

iii.) Apply a general optimizer, using the mixture's log-likelihood function.

what do we hope from this? better convergence proof of concept i.e. not complete failure

raise questions about implementation, clara fctn optim params

the subsequent chapter is devoted to answering this question by documenting the development of norMmix

Chapter 2

The `norMmix` Package

2.1 Introduction to the Package

For this thesis, an R package was developed that implements the algorithm that fits multivariate normal mixtures to given data. [1] There is a lot of unused code still in the package. These were at one point implemented used and discarded. They are still included for demonstration. The `norMmix` package is constructed around the `norMmix` object that codifies a `nor`mal `M`ultivariate `mix`ture model, and the `llnorMmix()` function, that calculates the log-likelihood given a model and data.

In the table 2.1 the notation used in code is listed along with a translation to the previously used mathematical notation. Additionally, functions with ambiguous names are listed here.

The package contains the following functionality:

The package relies on `optim` from the `stats` package for general optimization. we use the standard method implemented in `optim` which is `BFGS`, which is a quasi-Newton method (also known as a variable metric algorithm) as described in Broyden (1970) among others.

The workflow when using the package is as follows. The function `rnorMmix` can be used to generate data from a `norMmix` object. The `MW` objects provide ready made examples and

[1]The package was written with R version 3.6.1 (2019-07-05) last updated on 2019-10-22.

In Notation	In Code
π_i	`w, weights`
Σ	`Sigma`
μ	`mu`
K	`k`
dimension	`p, dim, dims`
components	`cl, components`
$\boldsymbol{\Sigma}$ model	`model`
`cluster`'s CLARA	`clara`
`mclust`'s hierarchical clustering	`mclVVV`
`mclust`'s `Mclust` fuction	`mclust`

Table 2.1: Translation Table: Mathematical Notation to R Code

norMmix `norMmix()` is the 'init-method' for `norMmix` objects. There exist `is.norMmix` `rnorMmix` and `dnorMmix` functions.

parametrization The main functions that handle reparametrization of models from and to $\boldsymbol{LDL}^\top$ decomposition are `nMm2par` and `par2nMm`, which are inverse to each other.

MLE The function `norMmixMLE` marries the main components of this package. It initializes a model and parametrizes it for use with `optim`

model choice Using `norMmixMLE`, the function fitnMm allows fitting of multiple models and components. Functions analyzing the output of this are also provided, e.g. `BIC` and `print` methods.

misc There are also various methods of generics, like `logLik, print, BIC, AIC` and `nobs` as well as various `print` methods.

example objects Following the paper of Marron and Wand (1992) various example objects are provided and used for study. They follow the naming convention: MW + dimension + number. For example `MW213` for the 13th model of dimension 2.

simulations A good portion of the package is designed with the study of simulations in mind. Therefore there are functions provided to study large collections of evaluated data. e.g `compplot`

objects of study and the `norMmix` function can be used to define normal mixtures from scratch. Of course, other data sets can be used for analysis. The following functions rely, however, on the `matrix` data structure. So dataframes must be converted beforehand and non numerical data is not accepted.

Given data, the functions that accept it for analysis are mainly `norMmixMLE` and `fitnMm`. The former performs model fit on data, and the latter performs model selection, by calling `norMmixMLE` for specified `k` and `model` vectors.

2.1.1 norMmixMLE

The core of `norMmixMLE` is the application of `optim` in conjunction with `llnorMmix` as function to be optimized. `llnorMmix` can be accessed directly, however, it needs a transposed dataset. As stated in section 1.6 the MLE implicitly performs initialization. There are two options for this initialization step. One is the CLARA clustering algoritm, with non-standard arguments. The standard arguments are somewhat historic in origin and were, at the time, chosen because of hardware limitations. The newer function, due to this thesis' advisor Martin Mächler, was designed to be a 'sensible' alternative, but should be subject to further scrutiny. It is reproduced here.

```
>     norMmix::ssClaraL

function (n, k, p)
pmin(n, pmax(40, round(10 * log(n))) + round(2 * k * pmax(1,
    log(n * p))))
<bytecode: 0x5ae5478>
<environment: namespace:norMmix>
```

It is dependent on the size and dimension of the dataset, as well as the demanded number of clusters. The alternative to CLARA is `mclust`'s hierarchical agglomerative clustering, which follows the work of Fraley (1998). The intention behind using `mclust`'s initialization function is to directly compare how much difference the initialization process makes.

The initialization stage does not yield a normal mixture. This requires a way to transform a clustering into a mixture. The method chosen in this package is to use an m-step from the EM-algorithm. Unlike the EM-algorithm, clustering algorithms like CLARA produce binary cluster membership results, whereas the component membership of EM is determined as a probability value between 0 and 1. This is resolved by interpreting the results as probability values which are either 0 or 1. These are then used as the τ_j as described in section 1.2. This m-step is also taken from the `mclust` package for reasons better explained in section 2.2. It has the advantage of being able to generate a mixture object with the correct covariance model.

This mixture object is still in human readable form and not the necessary parameter vector demanded by `optim`. So an application of the function `nMm2par` is carried out, resulting in a starting value for `optim`.

Due to the nature of the package the returned results are more than abundant. Not only is the fitted model returned but also everything produced by `optim` and the entire dataset. Here are listed the stucture the returned values:

```
>     data(fSMI.12, package="norMmix")
>     str(fSMI.12$nMm[3,3][[1]], max=2)
```

```
List of 6
 $ norMmix:List of 6
  ..$ mu    : num [1:20, 1:3] 15.9 30.7 36.2 21.8 753 ...
  ..$ Sigma : num [1:20, 1:20, 1:3] 0.358 0 0 0 0 ...
  ..$ weight: num [1:3] 0.219 0.419 0.362
  ..$ k     : int 3
  ..$ dim   : int 20
  ..$ model : chr "EEI"
  ..- attr(*, "name")= chr "model = EEI , clusters = 3"
  ..- attr(*, "class")= chr "norMmix"
 $ optr   :List of 5
  ..$ par        : num [1:82] 0.264 0.118 15.903 30.67 36.155 ...
  ..$ value      : num 7370
  ..$ counts     : Named int [1:2] 232 88
  .. ..- attr(*, "names")= chr [1:2] "function" "gradient"
  ..$ convergence: int 0
  ..$ message    : NULL
 $ npar   : int 82
 $ n      : int 141
 $ x      : num [1:141, 1:20] 16.1 15.7 15.7 16.1 16.6 ...
  ..- attr(*, "dimnames")=List of 2
 $ cond   : num 1.72
 - attr(*, "class")= chr "norMmixMLE"
```

Besides `mclust` the package also relies on a number of other packages for various tasks. Listed in no particular order: `cluster`, `MASS`, `mvtnorm`, `mclust`, `mixtools` and `sfsmisc`.

since mclust is one of the more popular packages implementing the EM algo, we employ a lot of functions from mclust, to keep things around EM as similar as possible.

also relies on `mixtools` package for random generating function `rnorMmix` using `rmvnorm`.

2.2 On The Development of norMmix

about Cholesky decomp as ldlt. has advantages: fast, parametrically parsimonious, can easily compute loglikelihood

One dead-end was the parametrization of the weights of a mixture using the `logit` function.

```
> logit <- function(e) {
+     stopifnot(is.numeric(e) ,all(e >= 0), all.equal(sum(e),1))
+     qlogis(e[-1L])
+ }
> logitinv <- function(e) {
+     if (length(e)==0) {return(c(1))}
+     stopifnot(is.numeric(e))
+     e<- plogis(e)
+     sp. <- sum(e)
+     w <- c((1-sp.), e)
+ }
```

This uses the logistical function `logis` to transform to reduce the number of weights from K to $K-1$. Much like `clr1`, given a list of weights `logit` will transform them and `logitinv` will correctly reverse the transformation. However, unlike `clr1`, it will not transform an arbitrary list of length $K-1$ into a valid weight parameter. For example:

```
> w <- runif(7); ret <- logitinv(w)
> ret
```

```
[1] -3.0619765  0.6618731  0.5259321  0.5327439  0.6684533  0.5201867  0.5809991
[8]  0.5717883
```

The issue here is that the last line of `logitinv`, which is necessary to sum to one, but results in a negative value in `ret[1]` which is not a valid weight. The underlying issue is that not every tuple in $\mathbb{R}^{K-1}$ is a result of `logit`.

The option to use `logit` is still an argument to `norMmixMLE` by specifying `trafo="logit"`, but it shouldn't be used.

Another issue during development cropped up during fitting of high dimensional data. We studied the dataset `SMI.12` from the package `copula`, Hofert, Kojadinovic, Maechler, and Yan (2018):

```
> data(SMI.12, package="copula")
> str(SMI.12)
```

```
 num [1:141, 1:20] 16.1 15.7 15.7 16.1 16.6 ...
 - attr(*, "dimnames")=List of 2
  ..$ : chr [1:141] "2011-09-09" "2011-09-12" "2011-09-13" "2011-09-14" ...
  ..$ : chr [1:20] "ABBN" "ATLN" "ADEN" "CSGN" ...
```

A consequence of high dimensions is that matrix multiplication is no longer very stable. As a result, the covariance matrices produced by our own implementation of the EM-algorithms m-step (`mstep.nMm`) were not positive definite. In the case of `SMI.12`, several

```
>     plot(MW215)
```

Figure 2.1: Demonstration of the MW Object `MW215`. Correct model: `model="VEE", k=3`

covariance matrices are degenerate, which results in cancellation error with near-zero entries. We attempted to correct this with the function `forcePositive`, which simply tries to set $\boldsymbol{D}$ in $\boldsymbol{LDL}^\top$ greater than zero. This didn't resolve the issue, since a non-negligible part of the numerical error was in the $\boldsymbol{L}$ matrix and the resultant covariance matrix was still not positive definite.

We eventually resolved this issue by abandoning our own implementation and using the functions from the `Mclust` package. Not only were these numerically stable they were also able to differentiate between models, whereas ours would assume VVV for every fit.

testing of mvtnorm as proof that ldlt is in fact faster parametrization

mention, that there may be faster ways to apply backsolve. quote knuth about premature optimization?

not possible to sensibly compare normal mixtures except maybe a strange sorting algorithm using Mahalanobis distance or Kullback-Leibler distance or similar (Hellinger), but not numerically sensible to integrate over potentially high-dimensional spaces.

2.3 Demonstration

To end this chapter, here a small demonstration of the capabilities of `norMmix`. First a small plot to show an MW mixture.

It is a trimodal mixture along the diagonal.

```
>     set.seed(2019); x <- rnorMmix(500, MW215)
>     system.time(mleResult <- norMmixMLE(x, 3, "VEE"))
```

```
initial  value 2206.907425
iter  10 value 2147.633703
iter  20 value 2125.658743
final  value 2125.658364
converged
   user  system elapsed
  0.260   0.012   0.271
```

```
>     mleResult
```

```
object of class 'norMmixMLE'
norMmix object:
multivariate normal mixture model with the following attributes:
name:                  model = VEE , components = 3
 model:                  VEE
 dimension:         2
 components:         3
weight of components 0.365 0.325 0.31

returned from optim:
```

```
function gradient
      75       22

log-likelihood: -2125.658

 nobs         npar         nobs/npar
 500           13           38.46154
```

261 Here are the results of a run of `norMmixMLE` and below the graphical display of the results.

```
>      op <- par(mfrow=c(1,2), mar=c(1,2,3,1))
>      plot(MW215, asp=1, ylab='', xlab='')
>      points(x, col=adjustcolor("black", 0.5))
>      plot(MW215, asp=1, ylab='', xlab='')
>      plot(mleResult, fillcolor=norMmix:::nMmcols[2], newWindow=FALSE, points=FALSE)
>      legend("bottomright", legend=c("correct", "fitted"),
+             fill=norMmix:::nMmcols[1:2])
>      par(op)
```

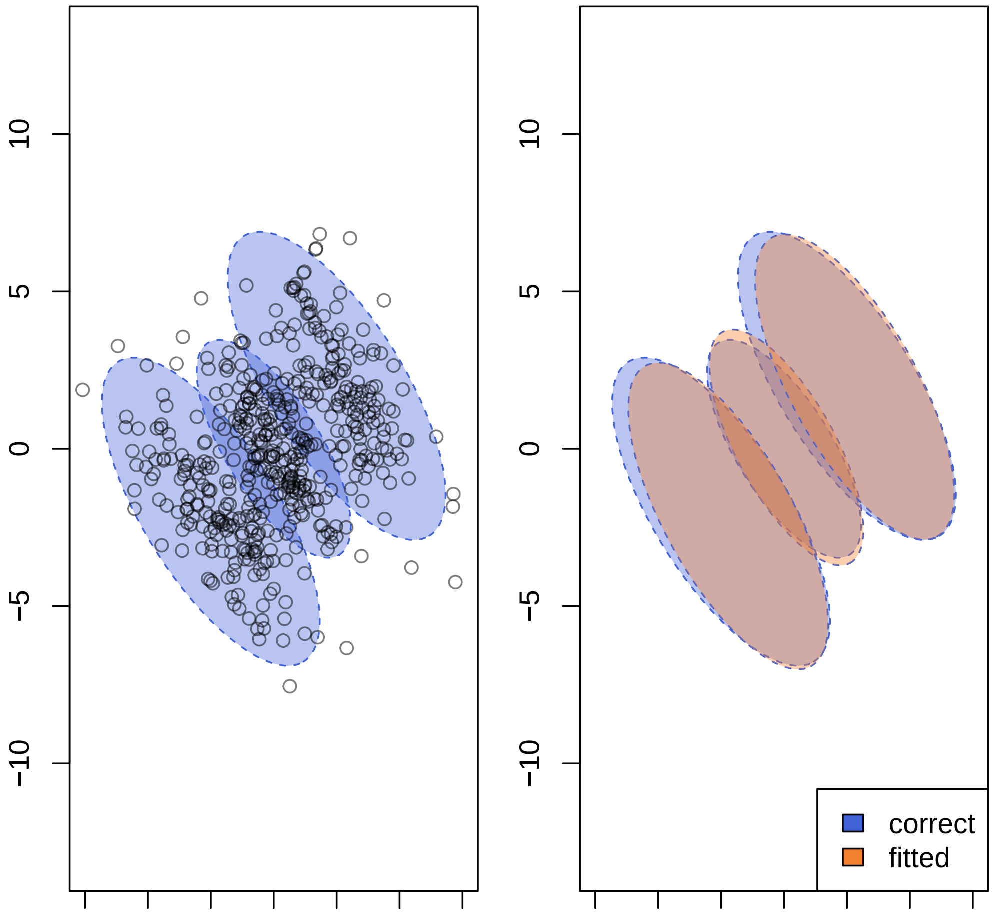


Figure 2.2: Correct Mixture (left) and Fitted overlayed in orange (right)

Chapter 3

Comparing Algorithms

With the `norMmix` package explained, we can turn to comparing it to existing methods. As previously stated, the implementation representing the EM-algorithm is the `mclust` package. It will be used with very little deviation from out-of-the-box, safe for restriction of the covariance models. This is done, so we can compare like with like. The specific command that performs the EM-algorithm is:

```
>     mclust::Mclust(x, G=cl, modelNames=mo)$BIC
```

Where `cl` is a vector of integers of however many components we are trying to fit and `mo` are the model names:

```
 [1] "EII" "VII" "EEI" "VEI" "EVI" "VVI" "EEE" "VEE" "EVV" "VVV"
```

The `$BIC` element of the results is taken as the main tool for model selection, as it is advertised in the package authors paper Scrucca et al. (2016).

There is however a small but crucial change applied to these results. The `mclust` package authors have flipped the definition of the BIC to mean:

$$2ln(\hat{L}) - ln(n)\#\{par\} \tag{3.0.0.1}$$

instead of the more common

$$ln(n)\#\{par\} - 2ln(\hat{L}) \tag{3.0.0.2}$$

Where n is the number of observations, $\#\{\text{par}\}$ is the cardinality of the parameter vector and $\hat{L}$ is the estimated log-likelihood.

So, even if not explicitly mentioned, we use the negative of the values returned by `mclust`.

Another thing that should be stated before all else is the difference in initialization between mclust's pre-clustering and CLARA. CLARA is dependent on random number generators (RNG). As such, unless a fixed seed is chosen, every iteration of CLARA will return a different result. Unlike `mclust`, which will, for given data, always return the same results. The effect on the following findings is that results will spread out for data obtained from CLARA results.

First, we illustrate the structure of the graphical results we will be presenting hereafter. The basic shape of the plots will be the BIC value plotted against the number of components. This is in line with `mclust`'s manner of visualizing data, however since our method is to some extent RNG dependent, we are forced to display multiple runs of the algorithm on the same graph. Therefore we split the plot according to covariance model, putting 10 models in 10 graphs in a plot. Here an example:

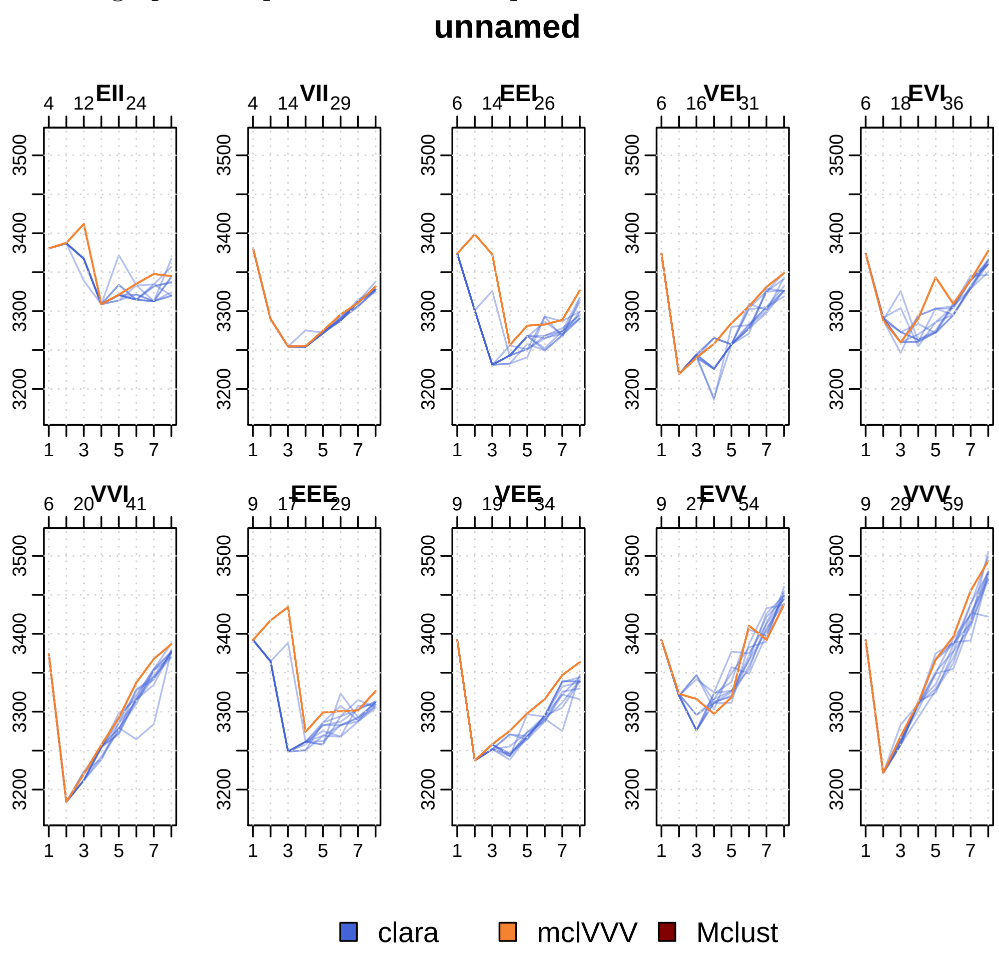


Figure 3.1: Example of Comparison Plot

As can be seen from the formula of the BIC value, lower is better. When selecting a model based on BIC, we take the model and component with the lowest value to be the best fitting model. Although this may not necessarily the 'correct' model, that is, the model from which the data arises.

There are many ways in which this type of model selection might miss the correct model, for example by 'gluing together' multiple components into one, or covering the dataset in a 'patchwork' of smaller components, to name a few.

We will discuss them as they arise in the following analysis of simulations

The simulations were set up very simply. An R script was written and in each the `norMmix` package is loaded, the datasets are defined and `fitnMm` was applied a number of times. An example script can be found in the appendix A.2.

A few things of interest are what happens:

- To time needed for the simulation

- When we vary the sample size of the data sets.
- When the generating mixture is 'difficult'.
- When the data does not arise from a normal mixture.

The data used here should have been provided along with this thesis in digital form in a folder called `/simulations`

3.1 Time Analysis

The data used here is taken from the subfolder `/simulations/2time`. From these, the system time was extracted and analyzed as can be gleaned from the following code. In it, we apply R 's `lm` function for fitting linear models to the times returned by the function call:

```
>     system.time(norMmixMLE(x, ...))[[1]]
```

We make here a choice that does not preserve any generality, as `system.time` produces more results, that could hold important information. However, since there is quite some measurement error to be expected as time approaches zero, we will content ourselves with lower expectations to the accuracy of the following results.

```
>     library(norMmix, lib.loc="~/ethz/BA/norMmix.Rcheck/")
>     # change this dir to whereever the simulations are saved
>     mainsav <- normalizePath("~/ethz/BA/Rscripts/")
>     savdir <- file.path(mainsav, "2time")
>     filelist <- list.files(savdir, pattern=".rds")
>     filelist <- grep("mcl.rds", filelist, invert=TRUE, value=TRUE)
>     f <- lapply(file.path(savdir,filelist), function(j) readRDS(j)$fit)
>     times <- unlist(lapply(f, function(j) extracttimes(j)[,,1]))
>     dims <- unlist(lapply(f, function(j) attr(extracttimes(j), "p")))
>     size <- unlist(lapply(f, function(j) attr(extracttimes(j), "n")))
>     ddims <- rep(dims, each=80)
>     ssize <- rep(size, each=80)
>     pars <- unlist(lapply(f, npar))
>     r <- lm(times ~ pars + ddims + ssize)
>     summary(r)
```

```
Call:
lm(formula = times ~ pars + ddims + ssize)

Residuals:
   Min      1Q Median      3Q     Max 
-86.89  -7.45  -1.55   6.30 556.32 

Coefficients:
              Estimate Std. Error t value Pr(>|t|)    
(Intercept) -1.727e+01  8.274e-01  -20.87   <2e-16 ***
pars         9.729e-01  1.056e-02   92.16   <2e-16 ***
ddims       -3.749e+00  2.216e-01  -16.92   <2e-16 ***
```

```
ssize         9.258e-03  3.887e-04   23.82   <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 21.57 on 7916 degrees of freedom
Multiple R-squared:  0.559,        Adjusted R-squared:  0.5588
F-statistic:  3344 on 3 and 7916 DF,  p-value: < 2.2e-16
```

The necessary time appears to be well explained by the parameter count. The purpose of this thesis is not to conduct complexity analysis, so we will leave it at this, satisfying our curiosity with a cursory look in figure 3.2, where we plot system time against parameter length.

We can see that time is almost one to one proportional to parameter length. It should be noted, that `MW51` is a very simple mixture. It is therefore sensible, that MLE should find an optimum faster.

```
>     plot(times~pars, log="xy", yaxt="n", xaxt="n", type="n")
>     legend("bottomright", legend=c("MW214", "MW34","MW51"),
+            fill=nMmcols[c(3,4,2)])
>     points(times[1:(80*30)]~pars[1:(80*30)],
+            log="xy", yaxt="n", xaxt="n", col=nMmcols[3])
>     points(times[(80*30+1):(80*60)]~pars[(80*30+1):(80*60)]
+            , log="xy", yaxt="n", xaxt="n", col=nMmcols[4])
>     points(times[(60*80+1):(80*90)]~pars[(60*80+1):(80*90)],
+            log="xy", yaxt="n", xaxt="n", col=nMmcols[2])
>     grid()
>     sfsmisc::eaxis(1)
>     sfsmisc::eaxis(2)
```

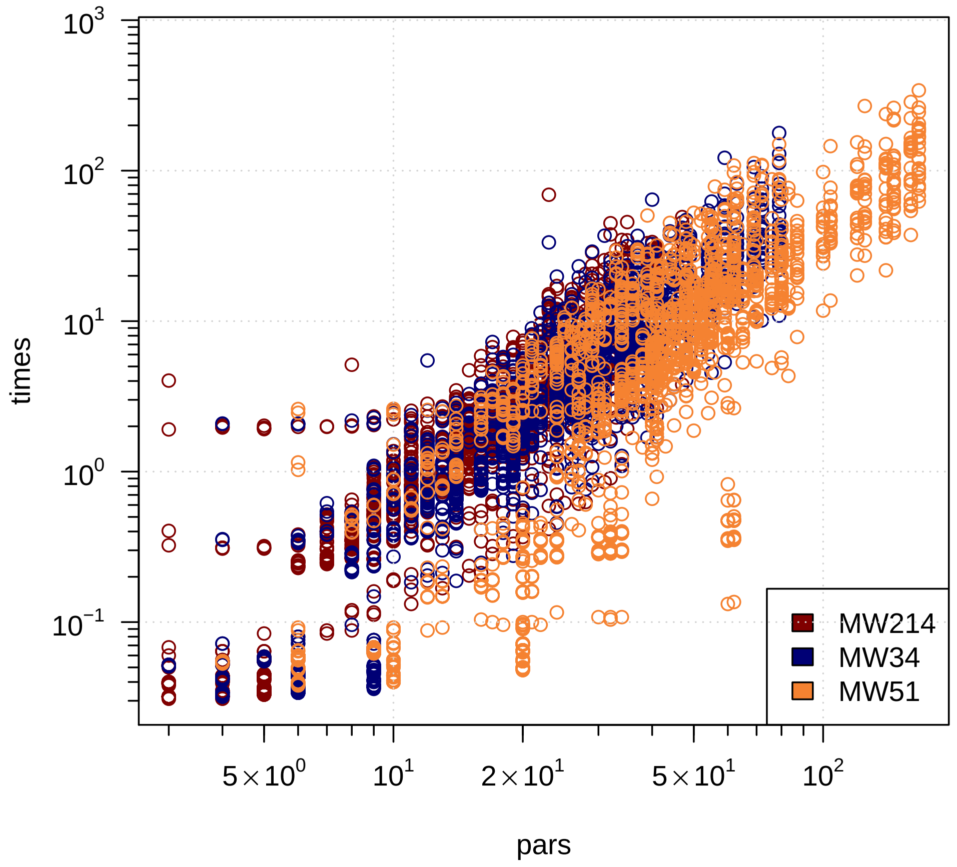


Figure 3.2: Log-log Plot of System Time against Parameter Length

3.2 Behaviour in `n`

What we would expect and like to see as we increase sample size, is a decrease in scattering of BIC values. To that end we again use simulation data `/2time`. In particular we show here the results of fitting to mixture model `MW34`, shown in figure 3.3. The graphs B.3 and B.4 show three columns of BIC plots, each representing different sample sizes, with $n = \{500, 1000, 2000\}$ respectively. Furthermore, the BIC values were divided by the samplesize, to normalize the values to an equal scale.

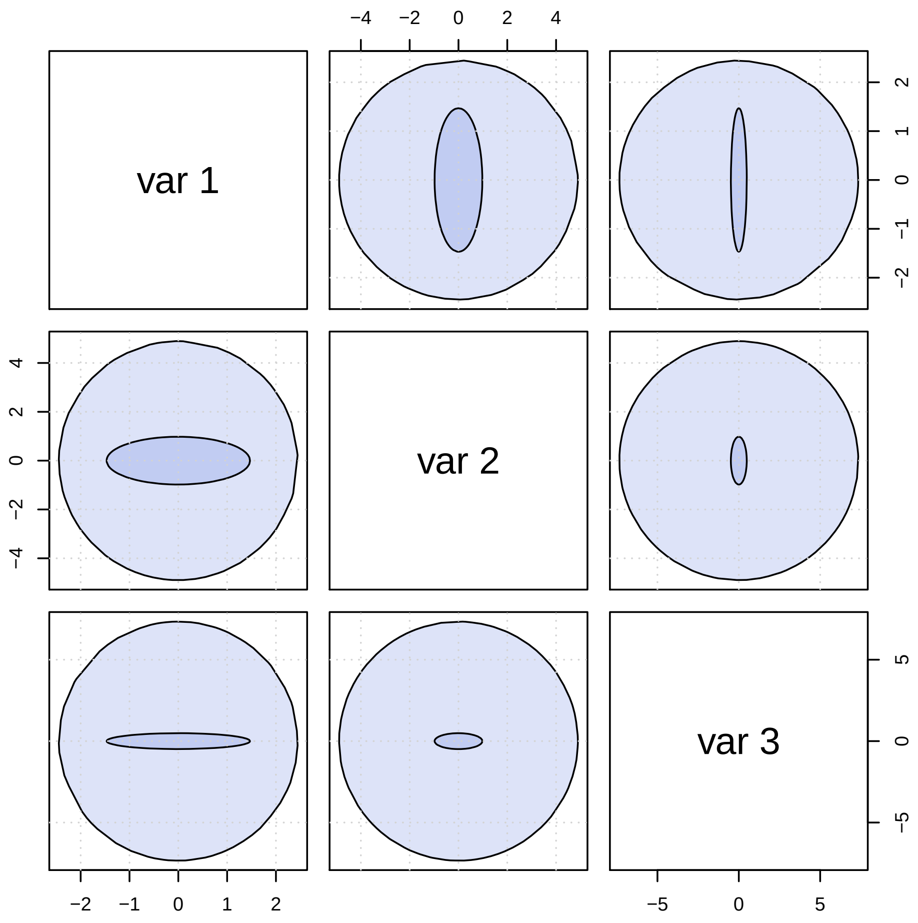


Figure 3.3: The mixture model `MW34`, a three dimensional, two component mixture with one smaller, lesser weighted component inside a smaller one.

As can be seen the desired effect is achieved. Of note are the behaviour of the model `VEI`, where the increase in observation corrects a selection error appearing at $n = 500$. Furthermore, the correct model `VVI` exhibits a very tight grouping. The instances where `mclust` is better than `norMmix` are quite infrequent.

This type of analysis was also conducted with mixture objects `MW214` and `MW51`, but were ommited due to the lack of clear results. They are provided in the appendix B.1, with brief discussions.

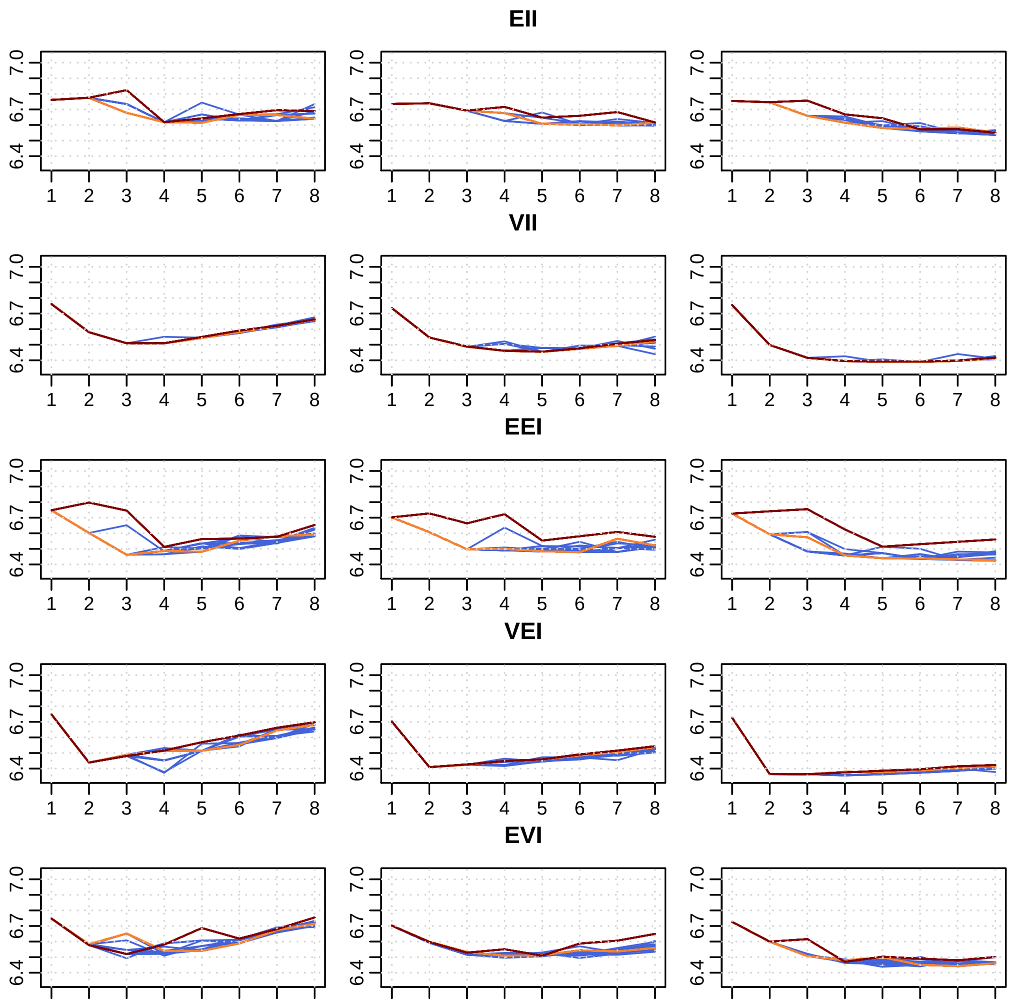


Figure 3.4: BIC values of MW34 with $n = 2000$

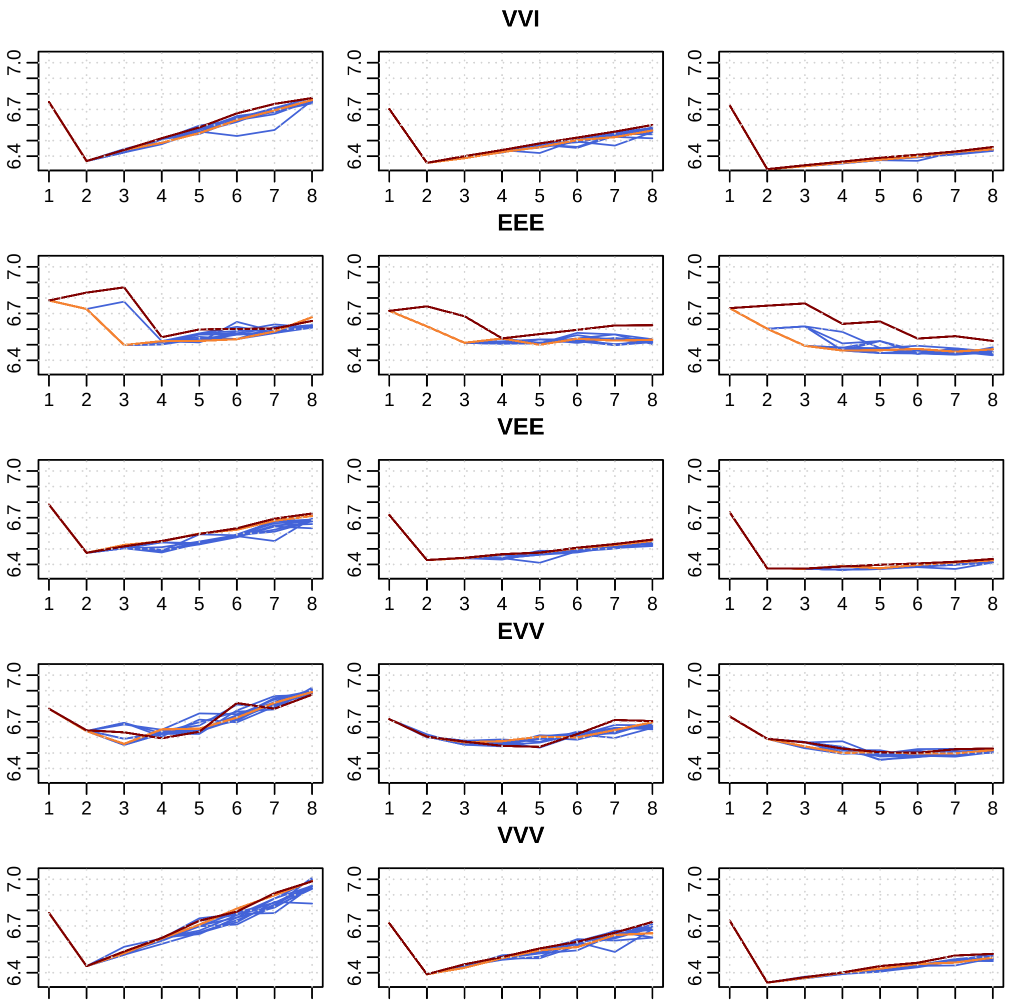


Figure 3.5: BIC values of `MW34` with $n = 2000$

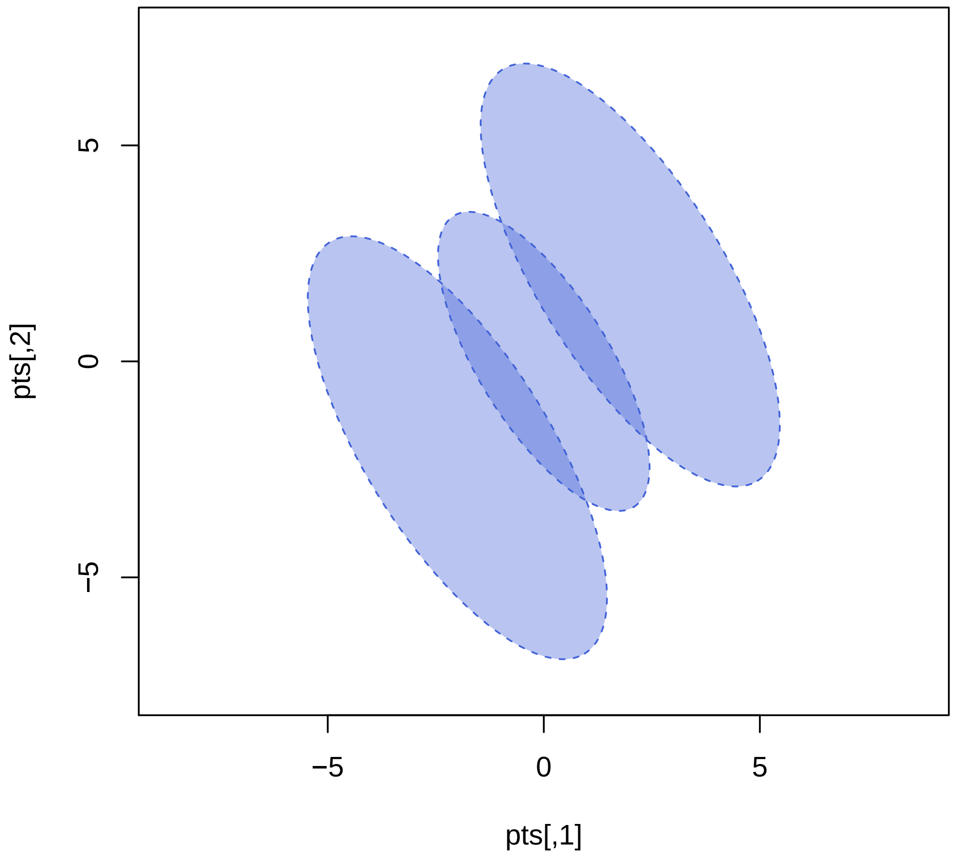


Figure 3.6: Trimodal mixture `MW215`. Three equally weighted, oriented, and shaped components of different volumes along the diagonal

3.3 Difficult Mixtures

In this section we analyze the two mixtures given by `MW215` and `MW214`. These are a trimodal and a claw-like distribution. These types of mixtures were also discussed in Marron and Wand (1992), in the univariate case, where they proved to be difficult to fit.

First the trimodal mixture shown in figure 3.6. The difficulty lies in the components of various sizes lying close together.

We can see, that in many cases both initialization methods `clara` and `mclVVV` manage to achieve a lower BIC value than `mclust`. Although in the case of the correct model and cluster, `k=3, model="VEE"` the three algorithms coincide.

A search for best values reveals, that the best models selected are in almost all cases the correct model.

```
     model   count
[1,] "2 VVI" "49"
[2,] "4 VEI" "1"
```

```
>      compplot(clarabic, mclbic, mclustbic, main="Fit of MW34")
```

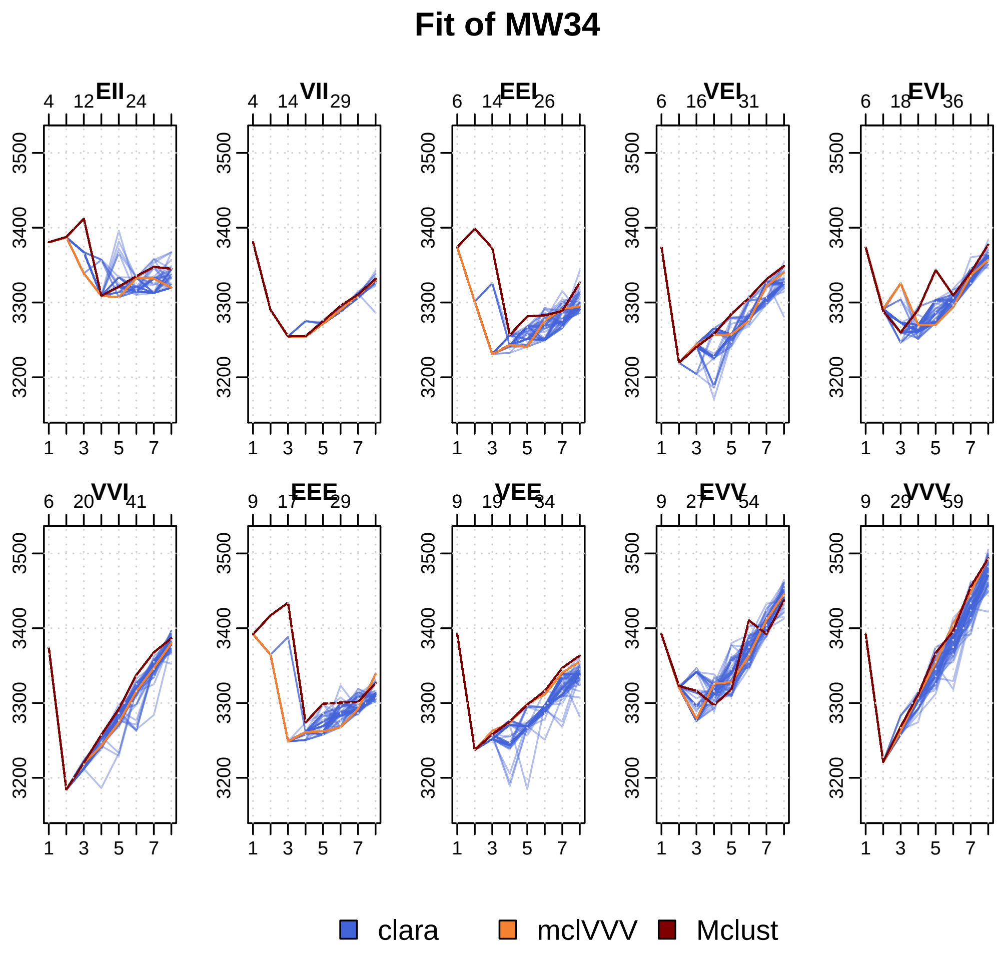
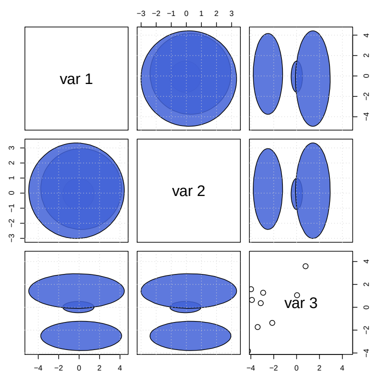
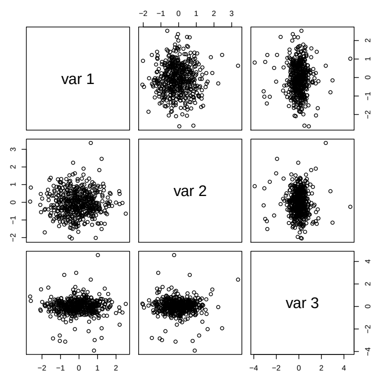


Figure 3.7: BIC values of `MW34`, correct: `model="VVI", k=2`

350 The one incorrect model looks like this:





351 and has the weights: 0.942, 0.0321, 0.0244, 0.002. This is an issue of spurious clusters.
352 These are clusters formed by a low number of datapoints conjoined into a component with
353 small determinant of its covariance matrix. It is a flaw in the `norMmix` package, that is
354 not adressed.

Now for the claw-like mixture, `MW214`. It is a mixture of six components and a very simple `"VII"` covariance model. A large encompassing component and five smaller, lightly wheighted components closely together along the diagonal. The inherent difficulty lies in the fact that the components overlap and are close together as well. It is shown in figure 3.8.

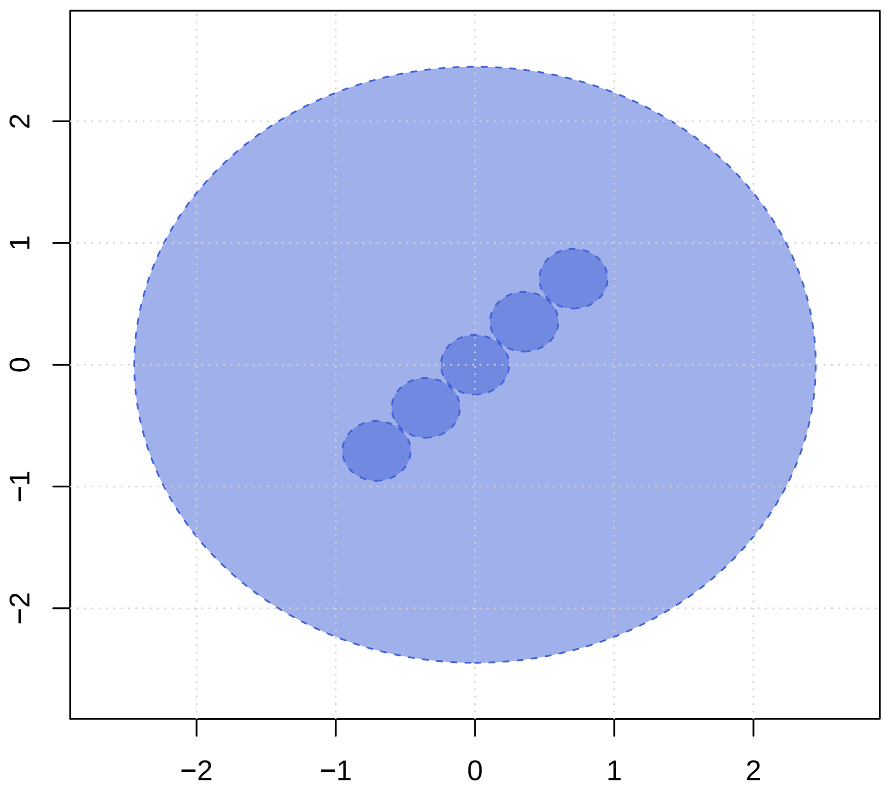


Figure 3.8: Claw-like mixture

We take a look at the best results per simulation again:

```
     model   count
[1,] "8 VII" "27"
[2,] "7 VEE" "8"
[3,] "7 VEI" "8"
[4,] "7 VII" "4"
[5,] "8 VEE" "3"
```

And here are the ten best values:

```
      comp model BIC
 [1,] "8"  "VEE" "1905.61014581771"
 [2,] "8"  "VEE" "1907.24944742008"
 [3,] "8"  "VEE" "1913.57109788463"
```

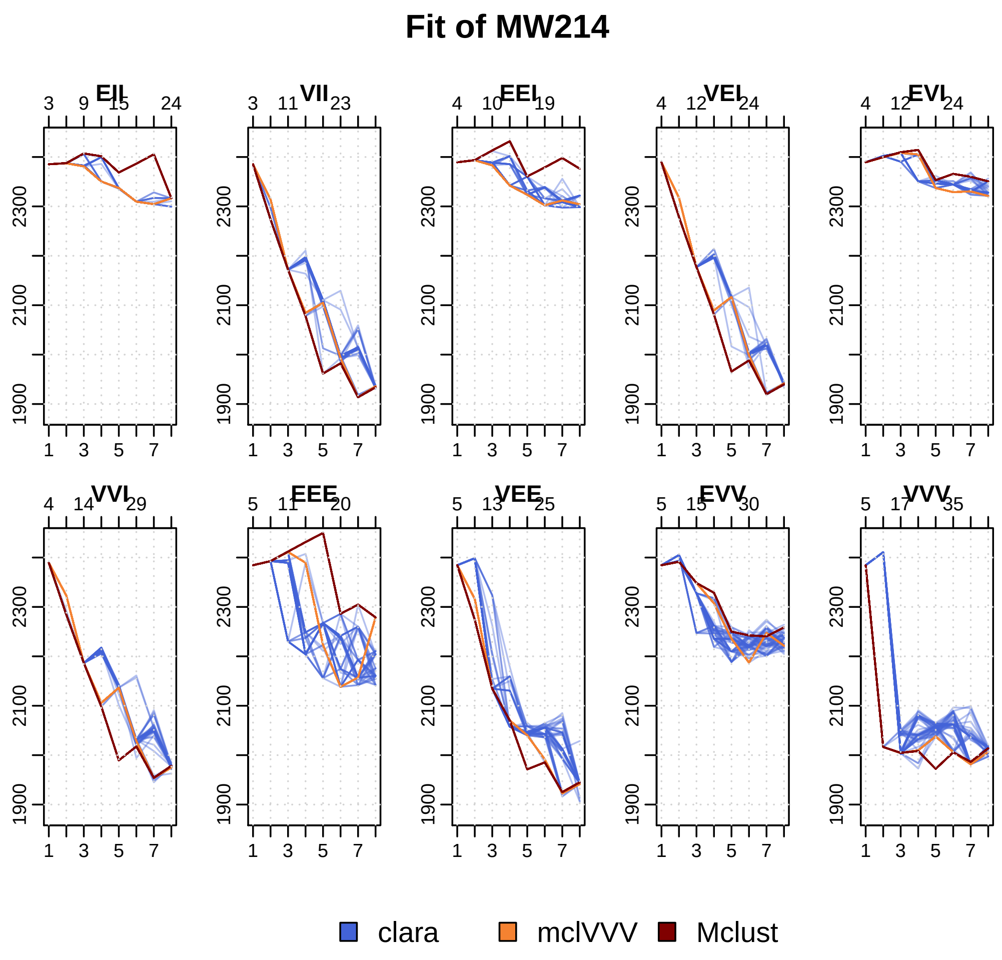


Figure 3.9: BIC values of claw-like mixture. Best fit: `model="VEE", k=8`, correct: model="VII", k=6

```
 [4,] "7"   "VII" "1913.68061849043"
 [5,] "7"   "VII" "1913.68062199219"
 [6,] "7"   "VEE" "1916.40190209225"
 [7,] "7"   "VEE" "1916.40195605402"
 [8,] "7"   "VEI" "1918.15484419568"
 [9,] "7"   "VII" "1918.35924550811"
[10,] "7"   "VII" "1918.4864952664"
```

Here some examples of fitted mixtures:

We can see, that, subtracting the obvious hiccups of the small erroneous components, `norMmix` has correctly found the 'intended' distribution. This is remarkable, given the small sample size and difficulty of distribution. As can be seen in figure 3.11, there are mistakes in the near best clusters, where the data is overlayed with a 'patchwork' of components.

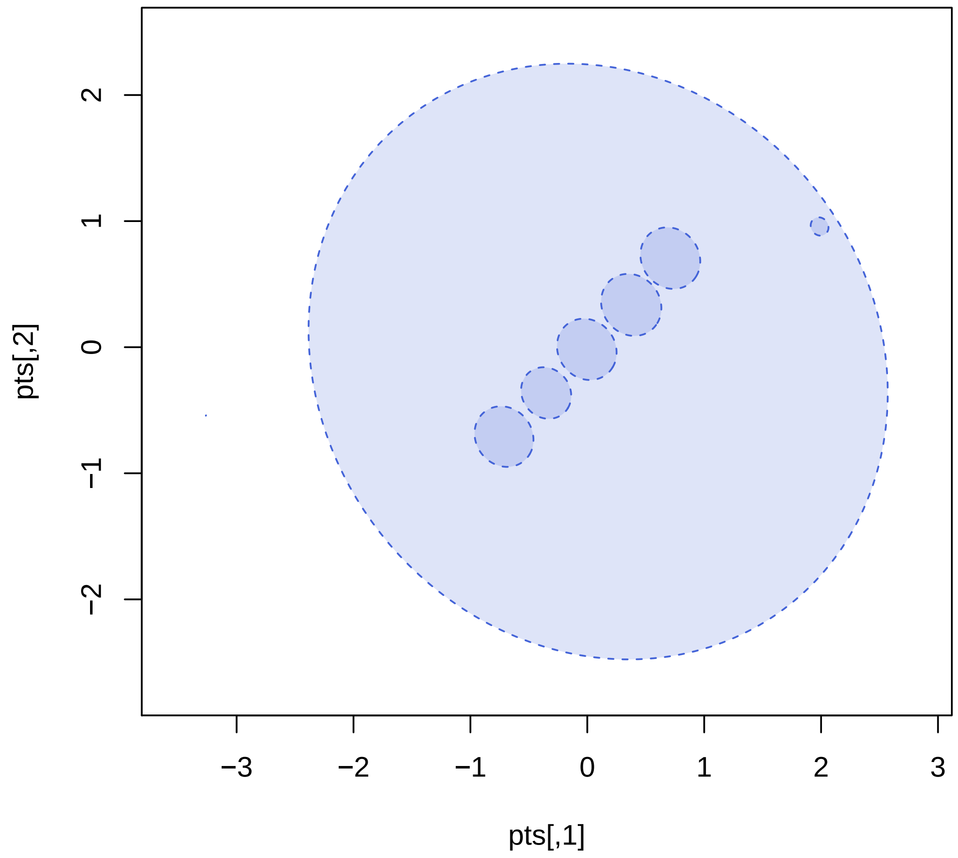


Figure 3.10: Best Fit over n = model selections. `model="VEE", k=8` Correct model `model="VII", k=6`. Of Note Here are the Spurious Clusters Appearing.

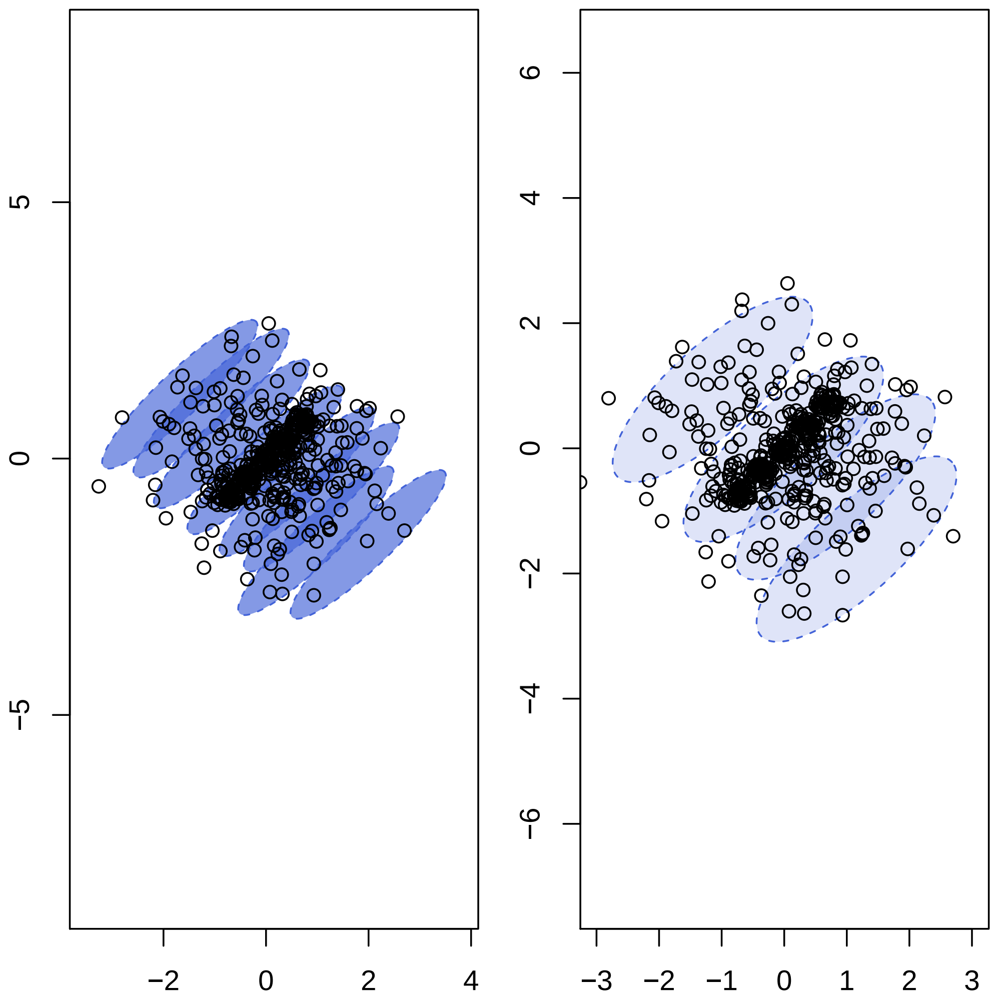


Figure 3.11: Two of the better clusters. They both follow the 'patchwork' covering strategy, laying patches of components over the dataset

3.4 Nonnormal Mixtures

here 2smi and 2var, maybe others as well. here mention that coverage of algo is extremely patchy. here 2smi: Using only datasets generated from the intended model can hide important structural errors in an algorithm. To that end we also applied `norMmix` to nonnormal data to see if any erratic behaviour appears.

The data used are the `SMI.12` and `loss` from the package `copula` Hofert et al. (2018), as well as the `iris` data included in base R .

We begin with the `SMI.12` dataset, that has already been discussed previously. This also doubles as high-dimensional analysis, as it is 20 dimensional.

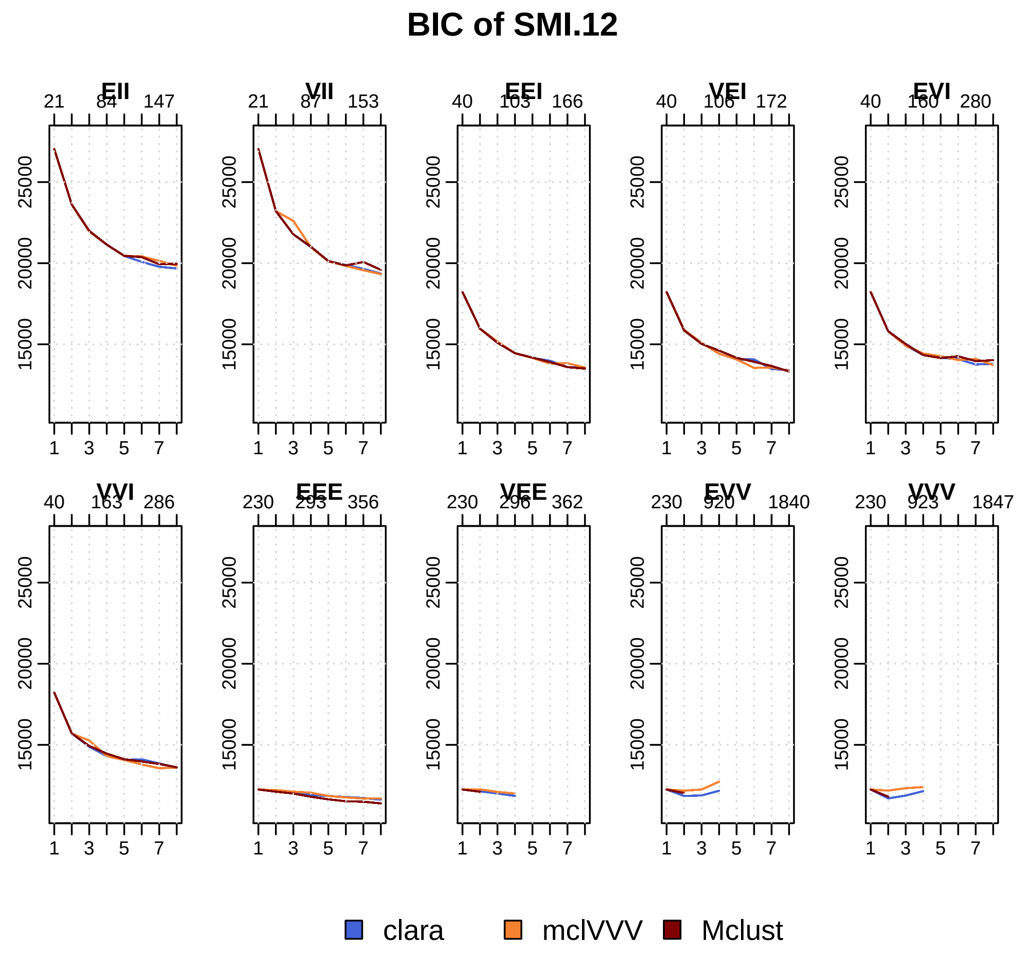


Figure 3.12: The BIC values of the `SMI.12` data. The blue line representing the `clara` values is covered by the other lines. The last three models are not plotted for all component sizes, as the algorithm returns an error if the fitting problem is ill defined.

While not very spectacular, the graphs show that even at large parameter counts our algorithm closes in on the same values as `mclust`. At these dimensions it is difficult to compare if these are actually equal, or even similar fits, but going by BIC values, it is at the very least equally viable as a working model. The last three models are not fully plotted for all components. The reason for this is that `norMmix` relies on `mclust` in its m-step. The `mclust` package halts computation when the clustering problem is badly posed. In this instance the problem is that the parameter count is much larger than the number of observations.

To illustrate, here are the parameter sizes for this simulation:

```
  EII VII EEI VEI EVI VVI EEE VEE  EVV  VVV
1  21  21  40  40  40  40 230 230  230  230
2  42  43  61  62  80  81 251 252  460  461
3  63  65  82  84 120 122 272 274  690  692
4  84  87 103 106 160 163 293 296  920  923
5 105 109 124 128 200 204 314 318 1150 1154
6 126 131 145 150 240 245 335 340 1380 1385
7 147 153 166 172 280 286 356 362 1610 1616
8 168 175 187 194 320 327 377 384 1840 1847
```

`SMI.12` has 141 observations, which is exceeded by the parameter count by all component sizes and covariance models. With a ratio of observations to parameters this low, it is desirable for clustering algorithms to break off and return an error, so conclusions are not drawn from ill posed problems.

For curiosity's sake we include here the system times taken for the simulations

```
   models
k     EII    VII    EEI    VEI     EVI    VVI     EEE     VEE      EVV      VVV
  1 0.059  0.051  0.058  0.059   0.070  0.070   0.201   0.202    0.273    0.275
  2 0.273  0.331  1.719  1.763   4.545  3.805  61.951  59.924  224.436  232.331
  3 0.435  1.950  4.816  5.248  12.660 12.860  96.099 125.053  660.375  638.954
  4 1.384  2.456  8.715  9.145  22.173 23.065 136.370 151.448 1438.264 1556.838
  5 1.869  3.289 13.293 14.703  26.584 28.580 218.786   0.683    0.690    0.682
  6 2.703  4.125 20.578 20.490  45.355 41.667 256.036   0.010    0.011    0.022
  7 2.235  4.337 31.705 34.893  89.809 83.015 353.466   0.012    0.014    0.025
  8 3.079 13.737 63.725 44.501 110.690 98.954 396.502   0.011    0.014    0.026
```

The longest, `model="VVV", k=4`, took 25.9473 minutes.

Next, we take a look at the `iris` dataset with 150 observations of 4 variables. The `fitnMm` was run with 25 different seeds. In this instance the `mclVVV` initialization was not applied, so we only compare to `mclust`.

The iris data originates from three types of plant species, which is not correctly identified by either `norMmix` or `mclust`. The best models chosen are:

```
     model   count
[1,] "6 VVV" "18"
[2,] "7 VVV" "7"
```

both far from three components. Furthermore `mclust` does not return values for some combinations of `k` and `model`. It is not clear what causes this, as a call to `Mclust` simply returns `NULL`.

Lastly, the data `loss`, from the `copula` package Hofert et al. (2018). This data is described as "Indemnity payment and allocated loss adjustment expense from an insurance company." It consists of 1500 observations with 4 variables. The BIC values are shown in 3.14

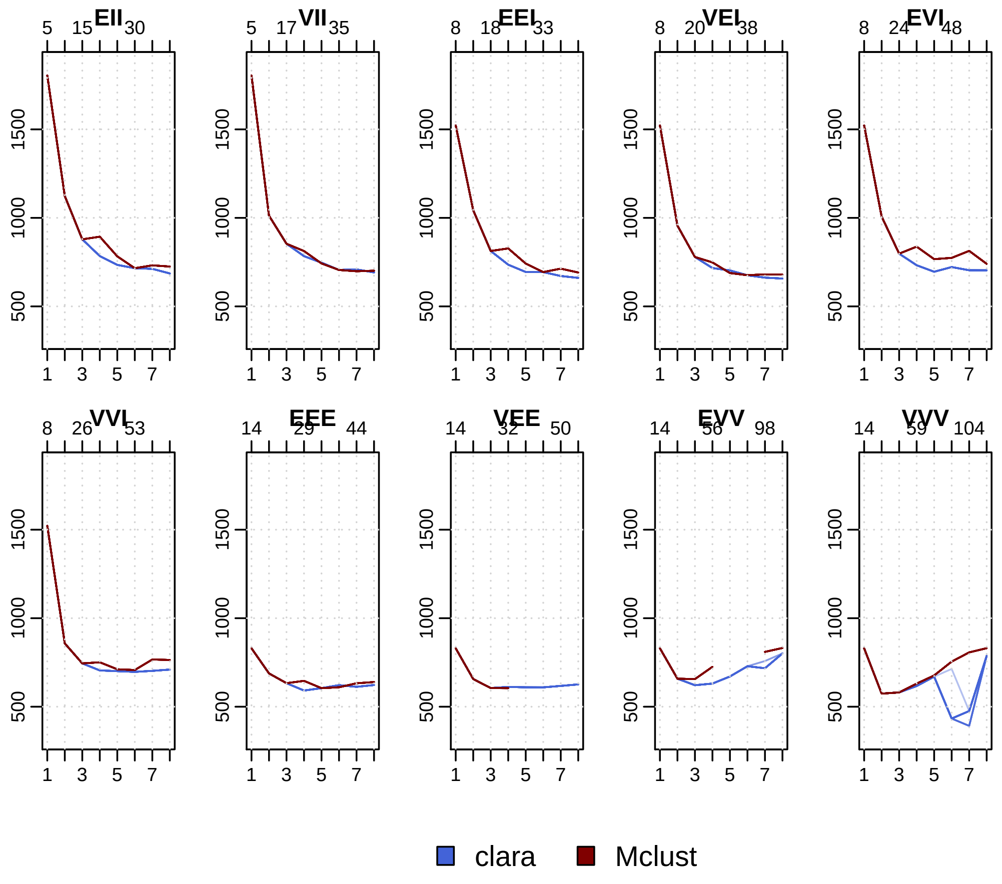


Figure 3.13: The BIC values for the `iris` data

403 The data resists any attempt at fitting. `mclust` returns `NULL`, as with `iris`. In `norMmix`,
404 the `optim` function encounters an error.

```
>     data(loss, package="copula")
>     to <- try(norMmixMLE(loss, k=3, model="EEI"))
```

```
initial  value 58082.835867
iter  10 value 56418.201189
iter  20 value 53014.372756
iter  30 value 49490.970255
iter  40 value 46802.853871
```

```
>     print(to)
```

```
[1] "Error in optim(initpar., neglogl, method = method, control = control) : \n  non-finite
attr(,"class")
[1] "try-error"
attr(,"condition")
<simpleError in optim(initpar., neglogl, method = method, control = control): non-finite fi
```

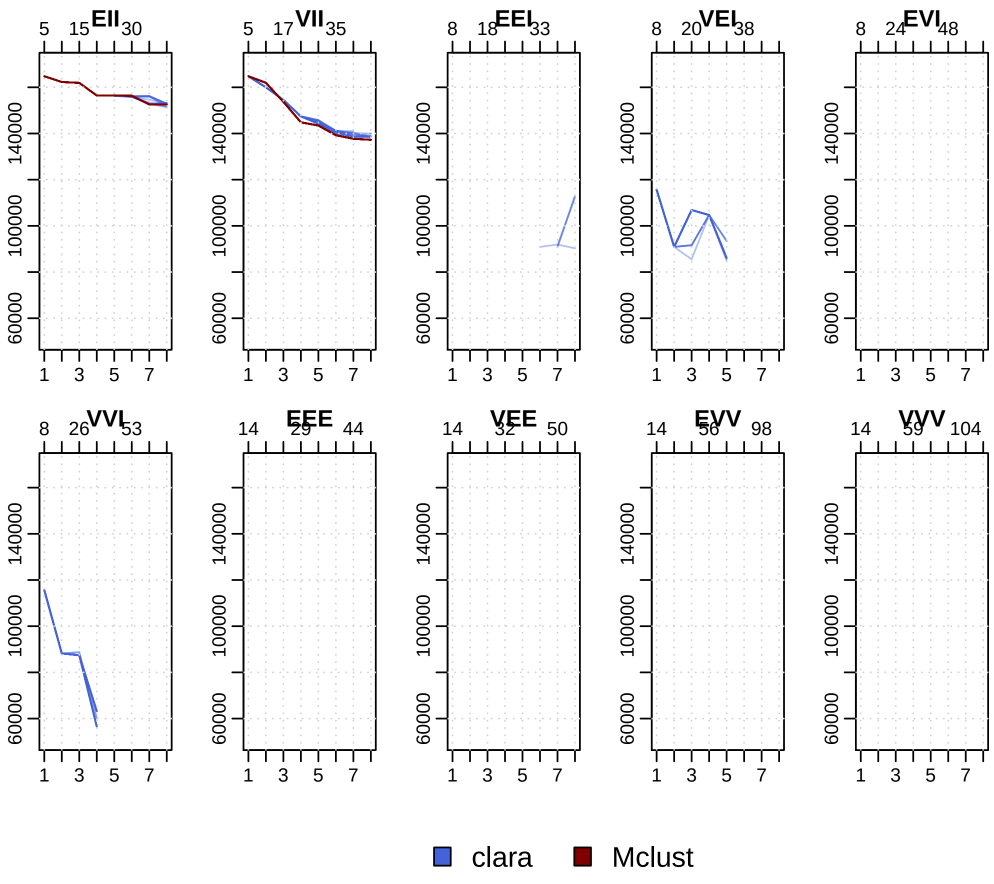


Figure 3.14: Loss data

Chapter 4

Discussion

one shortcoming is time inefficiency. largely due to implementation. proof of concept?? definitely possible to do model selection using a general optimizer.

As we have seen, the algorithm works and is in many cases equal if not better to existing clustering methods. The approach is also very generalizable, with the biggest hurdle being an efficient implementation of a log-likelihood function and a parametrization strategy. Should this approach be improved upon, it may provide a valuable tool in the arsenal of mixture model analysis.

There are many directions further research in this area may be conducted. For instance, the initialization methods may prove to be an essential factor in correct model selection. Furthermore, in the case of CLARA, the parameters chosen are somewhat arbitrary. It could yield useful results how CLARA behaves with different sampling parameters.

The investigation conducted in this thesis also falls short in the study of high-dimensional datasets. While we have looked into it with the analysis of the `SMI.12` data, the behaviour in these cases might also hold its own difficulties, that have not cropped up in the study of one dataset.

Further research could also go in the direction of model selection theory. The Bayesian Information Criterion was chosen in this work for its reliable results and usefulness, but other methods might yield more appropriate results.

There are also implementation related improvements, that could prove useful. For example, as seen in figure 3.10, spurious clusters are not accounted for at all in our implementation, which could strongly impact the strength of this tool. This is most likely the most pressing issue with the implementation in this package, that no measures against spurious clusters have been developed.

4.1 Acknowledgements

The Author would like to thank the 'Seminar fur Statistik' and ETH Zurich for providing the computing resources needed for the simulations used in this thesis.

Bibliography

Benaglia, T., D. Chauveau, D. R. Hunter, and D. Young (2009). mixtools: An R package for analyzing finite mixture models. *Journal of Statistical Software 32*(6), 1–29.

Broyden, C. G. (1970, 03). The Convergence of a Class of Double-rank Minimization Algorithms 1. General Considerations. *IMA Journal of Applied Mathematics 6*(1), 76–90.

Celeux, G. and G. Govaert (1995). Gaussian parsimonious clustering models. *Pattern Recognition 28*(5), 781–793.

Dempster, A., N. Laird, and D. Rubin (1977). Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society: Series B (Methodological) 39*(1), 1–22.

Fraley, C. (1998). Algorithms for model-based gaussian hierarchical clustering. *SIAM Journal on Scientific Computing 20*(1), 270–281.

Genz, A., F. Bretz, T. Miwa, X. Mi, F. Leisch, F. Scheipl, and T. Hothorn (2019). *mvtnorm: Multivariate Normal and t Distributions.* R package version 1.0-11.

Hofert, M., I. Kojadinovic, M. Maechler, and J. Yan (2018). *copula: Multivariate Dependence with Copulas.* R package version 0.999-19.1.

Maechler, M. (2019). *sfsmisc: Utilities from 'Seminar fuer Statistik' ETH Zurich.* R package version 1.1-4.

Maechler, M., P. Rousseeuw, A. Struyf, M. Hubert, and K. Hornik (2019). *cluster: Cluster Analysis Basics and Extensions.* R package version 2.1.0 — For new features, see the 'Changelog' file (in the package source).

Marron, S. and M. Wand (1992). Exact mean integrated squared error. *The Annals of Statistics 20*(2), 712–736.

McLachlan, G. and D. Peel (2000). *Finite Mixture Models* (1 ed.). Wiley Series in Probability and Statistics. Wiley-Interscience.

Pearson, K. and O. M. F. E. Henrici (1896). Vii. mathematical contributions to the theory of evolution. iii. regression, heredity, and panmixia. *Philosophical Transactions of the Royal Society of London. Series A, Containing Papers of a Mathematical or Physical Character 187*, 253–318.

Scrucca, L., M. Fop, B. Murphy, and A. Raftery (2016). mclust 5: Clustering, classification and density estimation using gaussian finite mixture models. *R Journal* (8(1)), 289–317.

Venables, W. N. and B. D. Ripley (2002). *Modern Applied Statistics with S* (Fourth ed.). New York: Springer. ISBN 0-387-95457-0.

Appendix A

R Code

A.1 `llnorMmix`

Here `llnorMmix`, since it is the central piece of the package, and 2time.R as an example of a simulation script.

```
#### the llnorMmix function, calculating log likelihood for a given
#### parameter vector

## Author: Nicolas Trutmann 2019-07-06

## Log-likelihood of parameter vector given data
#
# par:   parameter vector
# tx:    transposed sample matrix
# k:     number of components
# model: assumed distribution model of normal mixture
# trafo: either centered log ratio or logit
llnorMmix ← function(par, tx, k,
                      trafo=c("clr1", "logit"),
                      model=c("EII","VII","EEI","VEI","EVI",
                              "VVI","EEE","VEE","EVV","VVV")
                      ) {
    stopifnot(is.matrix(tx),
              length(k ← as.integer(k)) == 1, k >= 1)
    p ← nrow(tx)
#     x ← t(x) ## then only needed in   (x-mu[,i])^2  i=1..k

    # 2. transform

    model ← match.arg(model)
    trafo ← match.arg(trafo)

    l2pi ← log(2*pi)

    # 3. calc log-lik

    # get w

    w ← if (k==1) 1
        else switch(trafo,
                    "clr1" = clr1inv (par[1:(k-1)]),
                    "logit"= logitinv(par[1:(k-1)]),
                    stop("invalid 'trafo': ", trafo)
        )

    # start of relevant parameters:

```

```
    f ← k + p*k # weights -1 + means +1 => start of alpha
    # get mu
    mu ← matrix(par[k:(f-1L)], p,k)

    f1 ← f       # end of alpha if uniform
    f2 ← f+k-1L # end of alpha if var

    f1.1 ← f1 +1L # start of D. if alpha unif.
    f2.1 ← f1 + k # start of D. if alpha variable

    f11 ← f1 + p-1     # end of D. if D. uniform and alpha uniform
    f12 ← f1 +(p-1)*k # end     D. if D.   var   and alpha unif.
    f21 ← f2 + p-1     # end of D. if D. uniform and alpha variable
    f22 ← f2 +(p-1)*k # end of D. if D.   var   and alpha var.

    f11.1 ← f11 +1L # start of L if alpha unif  D unif
    f21.1 ← f21 +1L # start of L if alpha var   D unif
    f12.1 ← f12 +1L # start of L if alpha unif  D var
    f22.1 ← f22 +1L # start of L if alpha var   D var

    f111 ← f11 +   p*(p-1)/2 # end of L if alpha unif  D unif
    f211 ← f21 +   p*(p-1)/2 # end of L if alpha var   D unif
    f121 ← f12 + k*p*(p-1)/2 # end of L if alpha unif  D var
    f221 ← f22 + k*p*(p-1)/2 # end of L if alpha var   D var


    # initialize f(tx_i) i=1..n  vector of density values
    invl ← 0

    # calculate log-lik, see first case for explanation
    switch(model,
    "EII" = {
        alpha ← par[f]
        invalpha ← exp(-alpha)# = 1/exp(alpha)
        for (i in 1:k) {
            rss ← colSums(invalpha*(tx-mu[,i])^2)
            # this is vector of length n=sample size
            # calculates (tx-mu)t * Sigma^-1 * (tx-mu) for diagonal
            # cases.
            invl ← invl+w[i]*exp(-0.5*(p*(alpha+l2pi)+rss))
            # adds likelihood of one component to invl
            # the formula in exp() is the log of likelihood
            # still of length n
        }
    },
    # hereafter differences are difference in dimension in alpha and D.
    # alpha / alpha[i] and D. / D.[,i]

    "VII" = {
        alpha ← par[f:f2]
        for (i in 1:k) {
            rss ← colSums((tx-mu[,i])^2/exp(alpha[i]))
            invl ← invl+w[i]*exp(-0.5*(p*(alpha[i]+l2pi)+rss))
        }
    },

    "EEI" = {
        alpha ← par[f]
        D. ← par[f1.1:f11]
        D. ← c(-sum(D.),D.)
        D. ← D.-sum(D.)/p
        invD ← exp(alpha+D.)
        for (i in 1:k) {
            rss ← colSums((tx-mu[,i])^2/invD)
            invl ← invl+w[i]*exp(-0.5*(p*(alpha+l2pi)+rss))
        }
    },

    "VEI" = {
        alpha ← par[f:f2]
```

```
            D. ← par[f2.1:f21]
            D. ← c(-sum(D.), D.)
            D. ← D.-sum(D.)/p
            for (i in 1:k) {
                rss ← colSums((tx-mu[,i])^2/exp(alpha[i]+D.))
                invl ← invl+w[i]*exp(-0.5*(p*(alpha[i]+l2pi)+rss))
            }
        },

        "EVI" = {
            alpha ← par[f]
            D. ← matrix(par[f1.1:f12],p-1,k)
            D. ← apply(D.,2, function(j) c(-sum(j), j))
            D. ← apply(D.,2, function(j) j-sum(j)/p)
            for (i in 1:k) {
                rss ← colSums((tx-mu[,i])^2/exp(alpha+D.[,i]))
                invl ← invl+w[i]*exp(-0.5*(p*(alpha+l2pi)+rss))
            }
        },

        "VVI" = {
            alpha ← par[f:f2]
            D. ← matrix(par[f2.1:f22],p-1,k)
            D. ← apply(D.,2, function(j) c(-sum(j), j))
            D. ← apply(D.,2, function(j) j-sum(j)/p)
            for (i in 1:k) {
                rss ← colSums((tx-mu[,i])^2/exp(alpha[i]+D.[,i]))
                invl ← invl+w[i]*exp(-0.5*(p*(alpha[i]+l2pi)+rss))
            }
        },

        # here start the non-diagonal cases. main difference is the use
        # of backsolve() to calculate tx^t Sigma^-1 tx, works as follows:
        # assume Sigma = L D L^t, then Sigma^-1 = (L^t)^-1 D^-1 L^-1
        # y = L^-1 tx  => tx^t Sigma^-1 tx = y^t D^-1 y
        # y = backsolve(L., tx)

        "EEE" = {
            alpha ← par[f]
            D. ← par[f1.1:f11]
            D. ← c(-sum(D.), D.)
            D. ← D.-sum(D./p)
            invD ← exp(alpha+D.)
            L. ← diag(1,p)
            L.[lower.tri(L., diag=FALSE)] ← par[f11.1:f111]
            for (i in 1:k) {
                rss ← colSums(backsolve(L.,(tx-mu[,i]), upper.tri=FALSE)^2/invD)
                invl ← invl+w[i]*exp(-0.5*(p*(alpha+l2pi)+rss))
            }
        },

        "VEE" = {
            alpha ← par[f:f2]
            D. ← par[f2.1:f21]
            D. ← c(-sum(D.), D.)
            D. ← D.-sum(D./p)
            L. ← diag(1,p)
            L.[lower.tri(L., diag=FALSE)] ← par[f21.1:f211]
            for (i in 1:k) {
                rss ← colSums(backsolve(L., (tx-mu[,i]), upper.tri=FALSE)^2/exp(alpha
                    [i]+D.))
                invl ← invl+w[i]*exp(-0.5*(p*(alpha[i]+l2pi)+rss))
            }
        },

        "EVV" = {
            alpha ← par[f]
            D. ← matrix(par[f1.1:f12],p-1,k)
            D. ← apply(D.,2, function(j) c(-sum(j), j))
            D. ← apply(D.,2, function(j) j-sum(j)/p)
```

```
          L.temp ← matrix(par[f12.1:f121],p*(p-1)/2,k)
          for (i in 1:k) {
              L. ← diag(1,p)
              L.[lower.tri(L., diag=FALSE)] ← L.temp[,i]
              rss ← colSums(backsolve(L., (tx-mu[,i]), upper.tri=FALSE)^2/exp(alpha
                  +D.[,i]))
              invl ← invl+w[i]*exp(-0.5*(p*(alpha+l2pi)+rss))
          }
      },

      "VVV" = {
          alpha ← par[f:f2]
          D. ← matrix(par[f2.1:f22],p-1,k)
          D. ← apply(D.,2, function(j) c(-sum(j), j))
          D. ← apply(D.,2, function(j) j-sum(j)/p)
          invalpha ← exp(rep(alpha, each=p)+D.)
          L.temp ← matrix(par[f22.1:f221],p*(p-1)/2,k)
          L. ← diag(1,p)
          for (i in 1:k) {
              L.[lower.tri(L., diag=FALSE)] ← L.temp[,i]
              rss ← colSums(backsolve(L., (tx-mu[,i]), upper.tri=FALSE)^2/invalpha
                  [,i])
              invl ← invl+w[i]*exp(-0.5*(p*(alpha[i]+l2pi)+rss))
          }
      },
      ## otherwise
      stop("invalid model:", model)
      )

      ## return  sum_{i=1}^n log( f(tx_i) ) :
      sum(log(invl))
}
```

A.2 Example Simulation Script

here e.g. 2init.R and write some remarks on it.

```
## Intent: analyse time as function of p,k,n

nmmdir ← normalizePath("~/BachelorArbeit/norMmix.Rcheck/")
savdir ← normalizePath("~/BachelorArbeit/Rscripts/2time")
stopifnot(dir.exists(nmmdir), dir.exists(savdir))
library(norMmix, lib.loc=nmmdir)
library(mclust)

## at n=500,p=2 can do about 250xfitnMm(x,1:10) in 24h
seeds ← 1:10
sizes ← c(500, 1000, 2000)
nmm ← list(MW214, MW34, MW51)
## => about 100 cases

# for naming purposes
nmnames ← c("MW214", "MW34", "MW51")
sizenames ← c("500", "1000", "2000")
files ← vector(mode="character")

for (nm in 1:3) {
    for (size in sizes) {
    set.seed(2019); x ← rnorMmix(size, nmm[[nm]])
        for (seed in seeds) {
            set.seed(2019+seed)
            r ← tryCatch(fitnMm(x, k=1:8,
                                         optREPORT=1e4, maxit=1e4),
                           error = identity)
            filename ← sprintf("%s_size=%0.4d_seed=%0.2d.rds",
                                        nmnames[nm], size, seed)
            files ← append(files, filename)
            cat("===> saving to file:", filename, "\n")
            saveRDS(list(fit=r), file=file.path(savdir, filename))
        }
    }
}

fillis ← list()
for (i in seq_along(sizes)) {
    for (j in seq_along(nmnames)) {
        # for lack of AND matching, OR match everything else and invert
        ret ← grep(paste(sizenames[-i], nmnames[-j], sep="|"),
                    files, value=TRUE, invert=TRUE)
        fillis[[paste0(sizenames[i], nmnames[j])]] ← ret
    }
}

epfl(fillis, savdir)
```

Appendix B

Further Plots

```
>     library(norMmix, lib.loc="~/ethz/BA/norMmix.Rcheck/")
>     mainsav <- normalizePath("~/ethz/BA/Rscripts/")
```

B.1 Behaviour in n

Here are the further plots ommited in section 3.2. First is a very difficult mixture, ommited because it is studied in greater detail in section 1.1. Second is a very easy mixture, because all fitting lines overlap, making meaningful analysis futile.

MW214

```
>     MW214
```

```
norMmix object:
multivariate normal mixture model with the following attributes:
name:                  #14 Smooth Comb
 model:                  VII
 dimension:        2
 components:        6
weight of components 0.5 0.1 0.1 0.1 0.1 0.1
```

MW51

B.2 Unused Data

Unfortunately not all simulations were well planned. In part, because they were done for exploratory purposes.

Some are displayed here

smallinit:

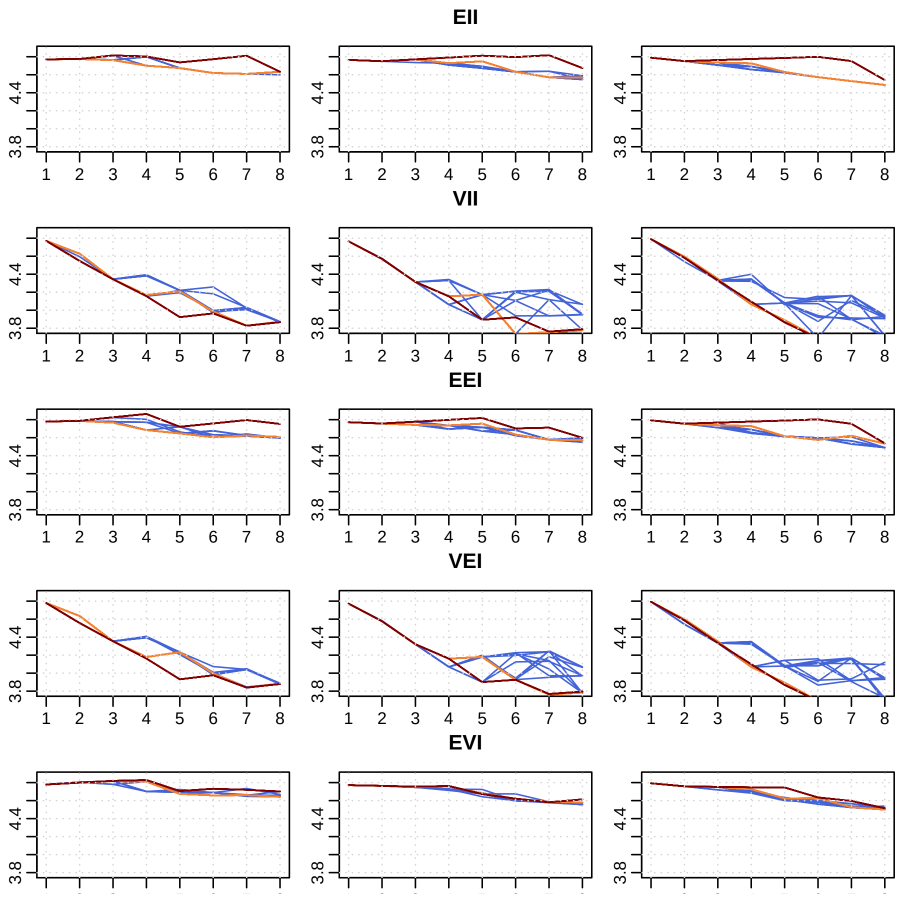


Figure B.1: BIC values of `MW34` with $n = 2000$

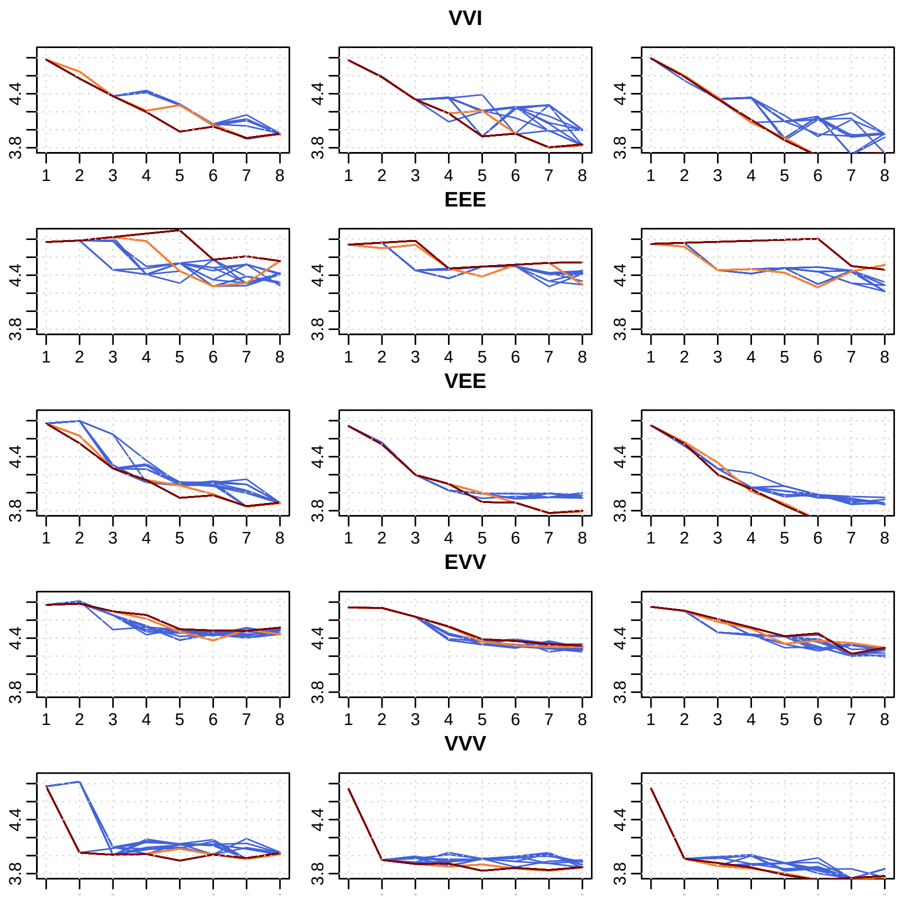


Figure B.2: BIC values of `MW34` with $n = 2000$

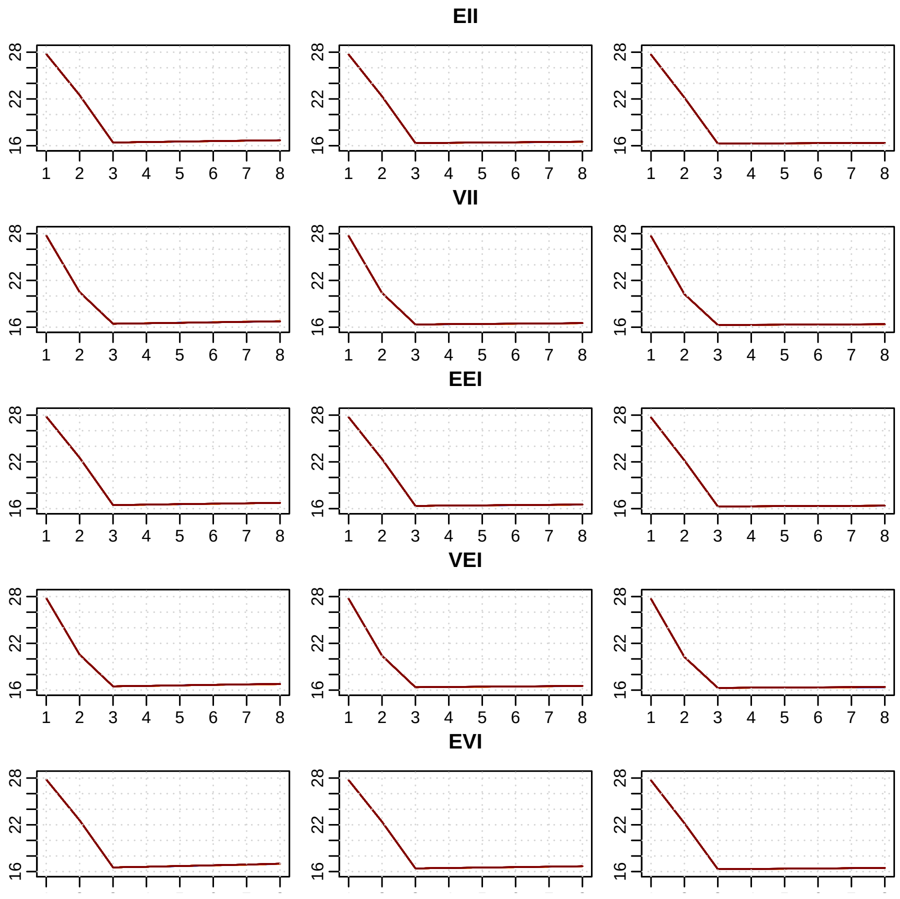


Figure B.3: BIC values of `MW51` with $n = 2000$

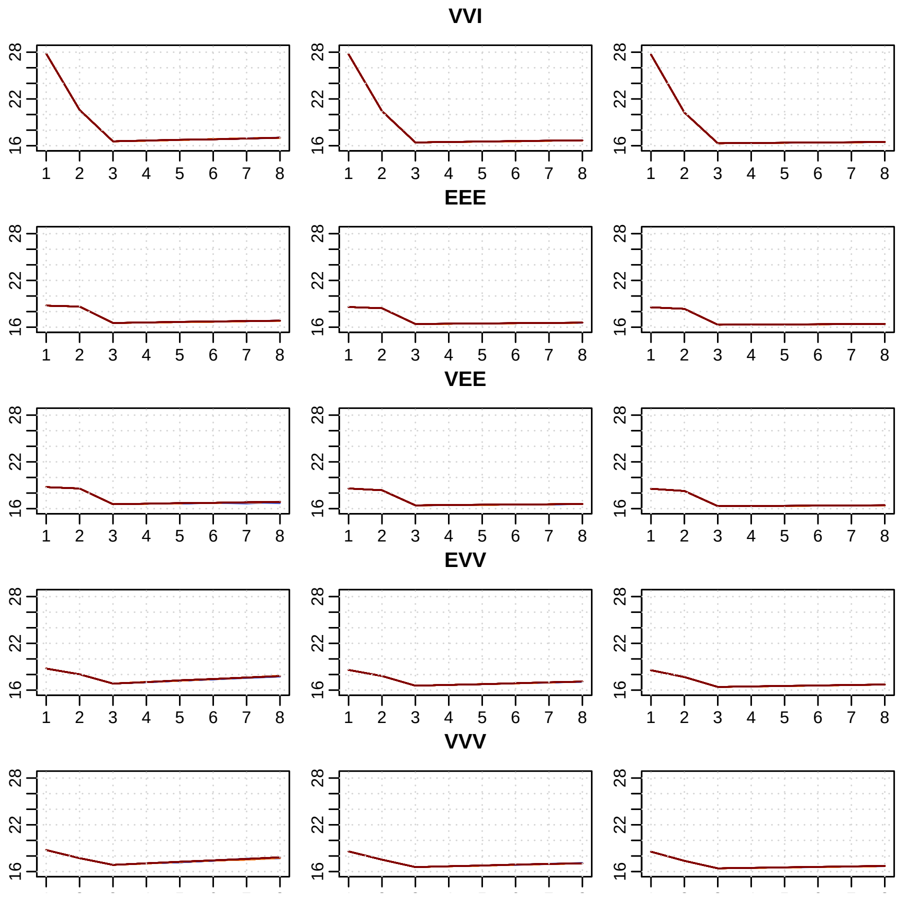
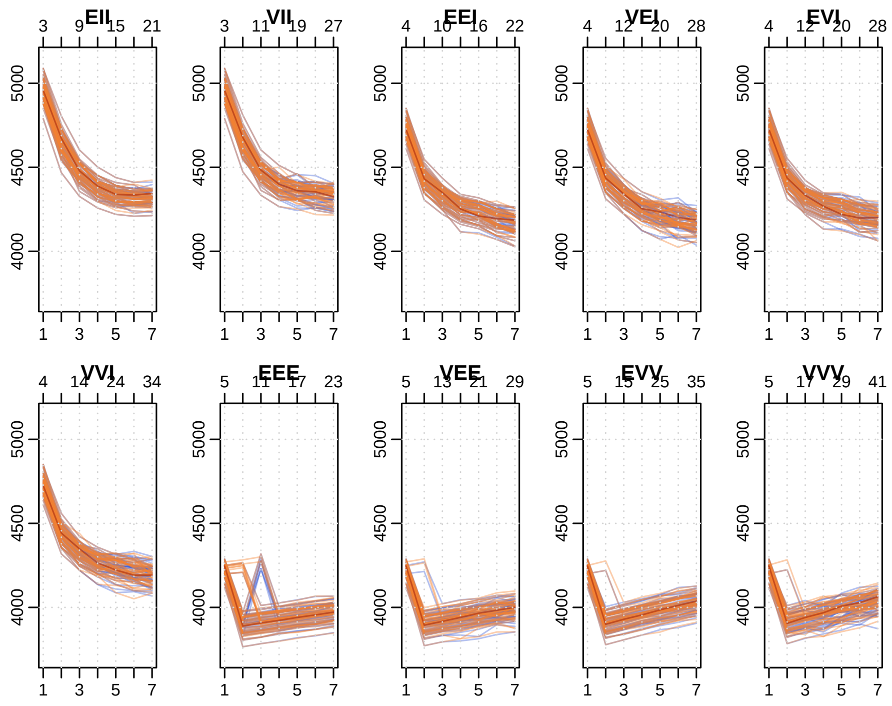
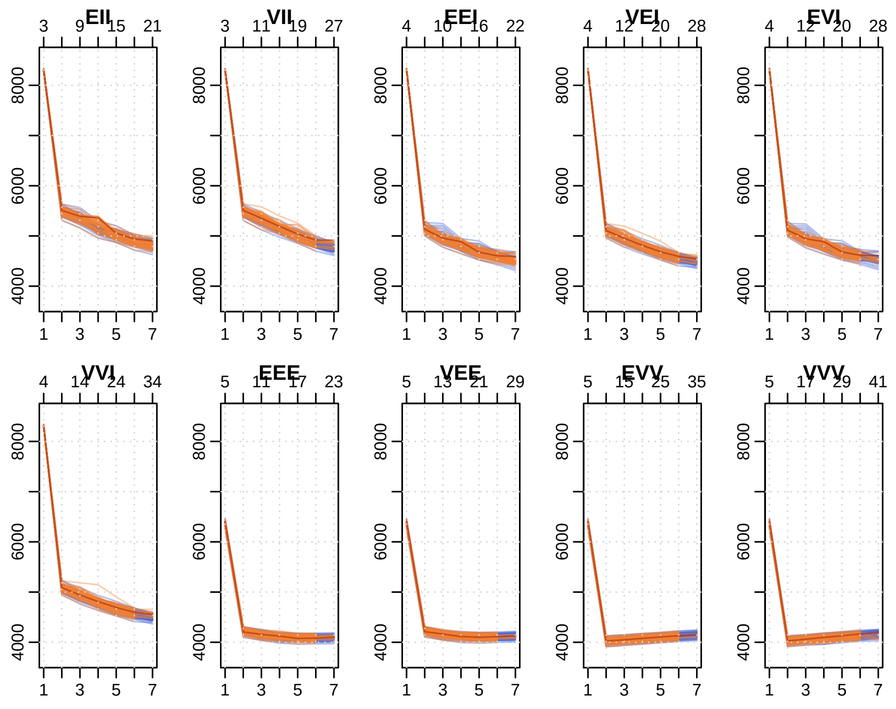
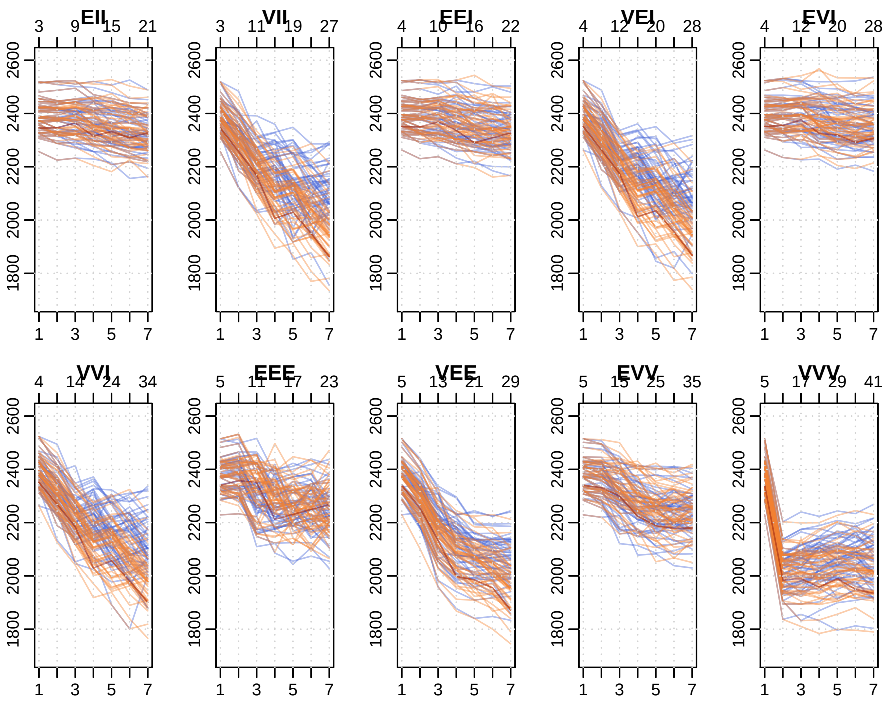


Figure B.4: BIC values of `MW34` with $n = 2000$

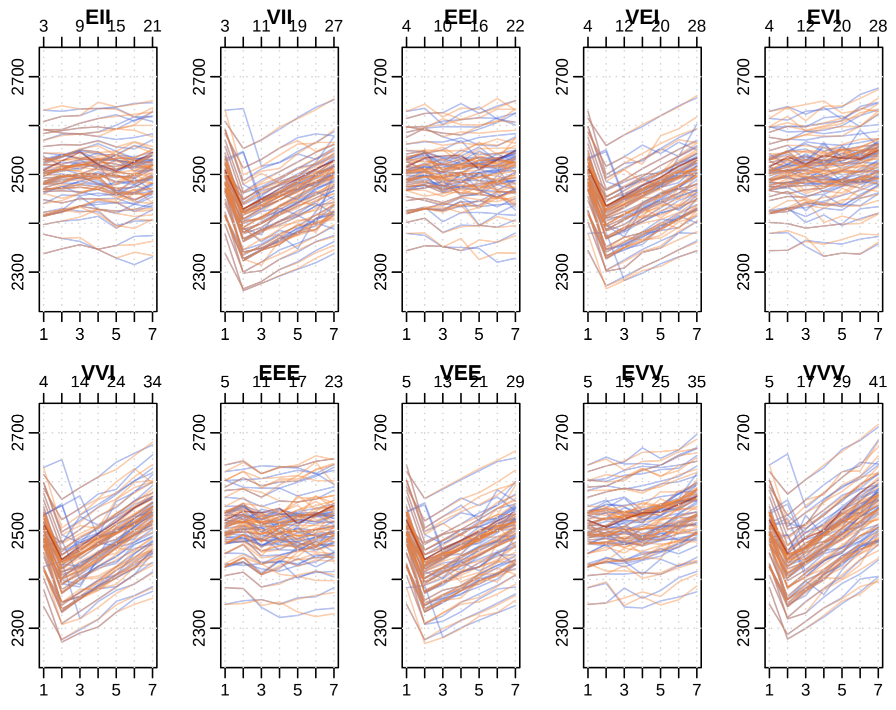

EII
VII
EEI
VEI
EVI
VVI
EEE
VEE
EVV
VVV
5000
4500
4000
1
3
5
7
🟦 clara
🟧 mclVVV
🟥 Mclust



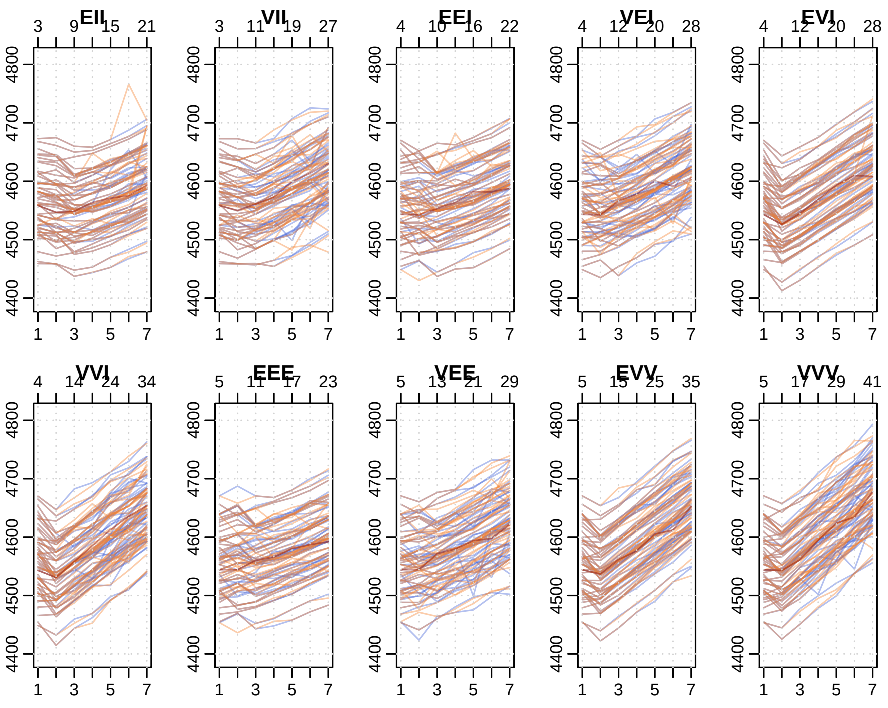
🟦 clara 🟧 mclVVV 🟥 Mclust



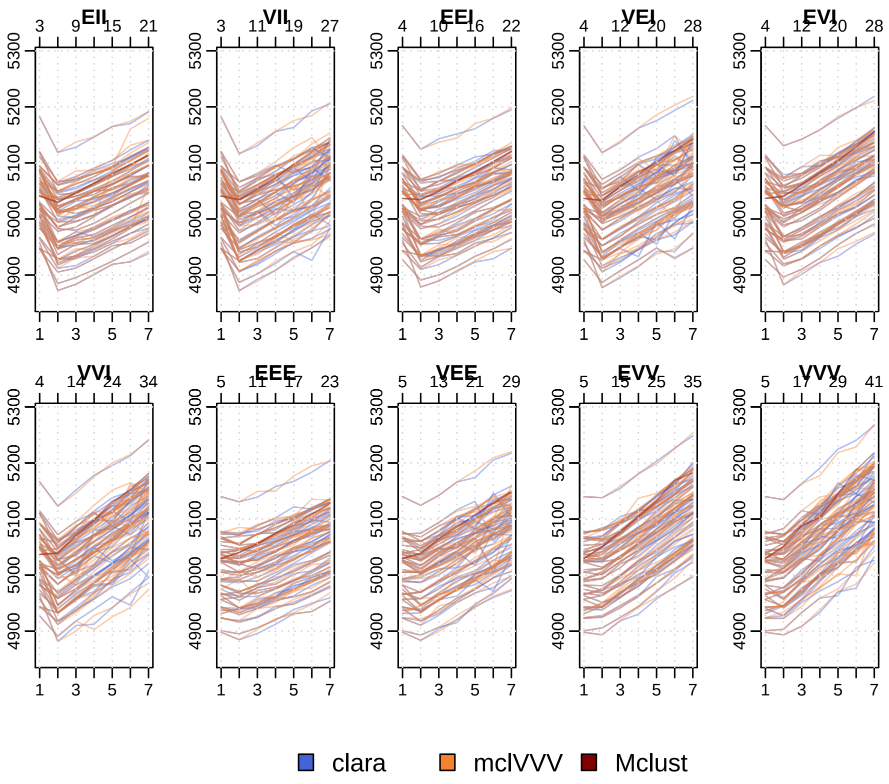
🟦 clara 🟧 mclVVV 🟥 Mclust

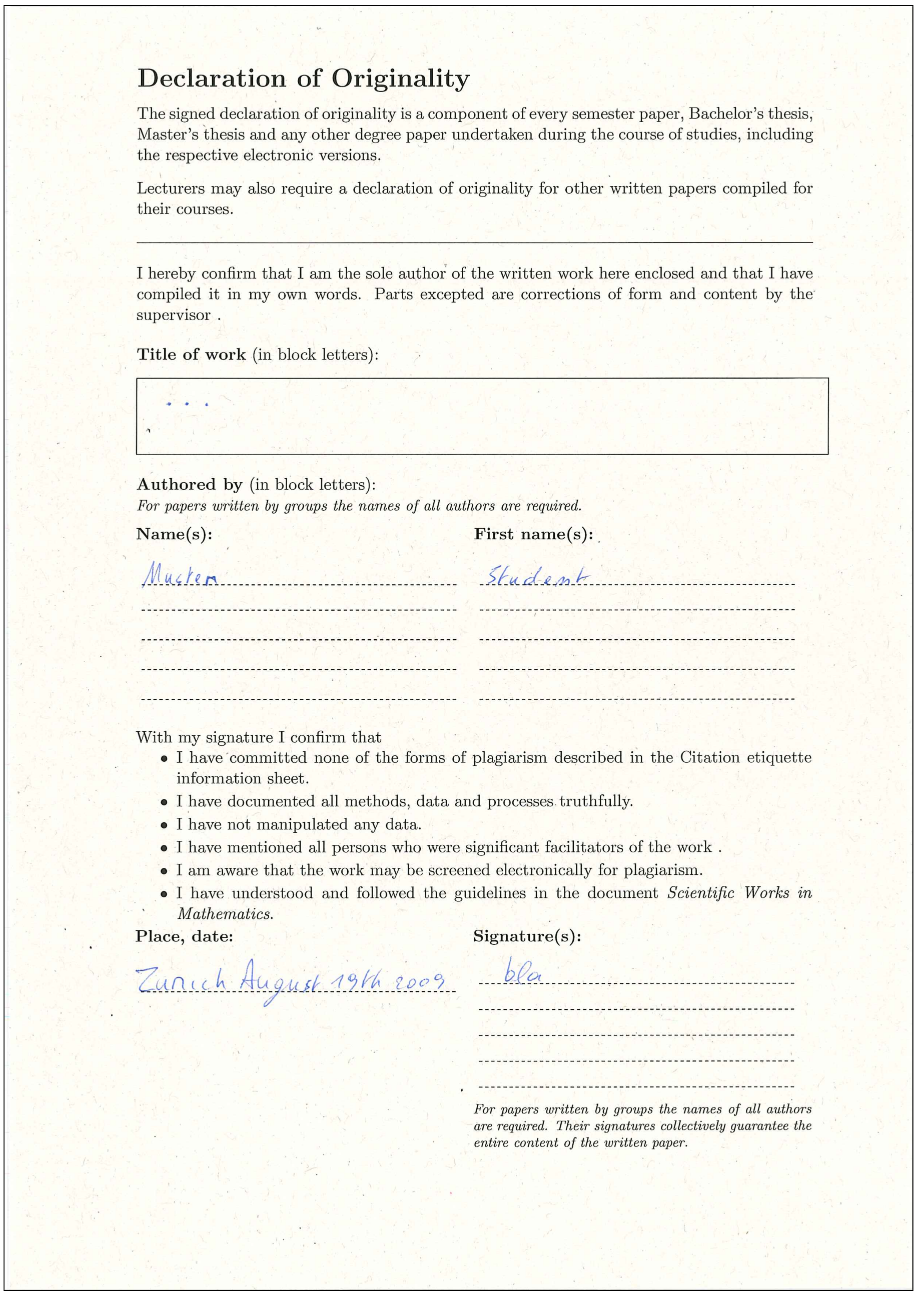


🟦 clara 🟧 mclVVV 🟥 Mclust



🟦 clara 🟧 mclVVV 🟥 Mclust


EII
VII
EEI
VEI
EVI
VVI
EEE
VEE
EVV
VVV
4900
5000
5100
5200
5300
🟦 clara
🟧 mclVVV
🟥 Mclust



Declaration of Originality

The signed declaration of originality is a component of every semester paper, Bachelor's thesis, Master's thesis and any other degree paper undertaken during the course of studies, including the respective electronic versions.

Lecturers may also require a declaration of originality for other written papers compiled for their courses.

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor .

Title of work (in block letters):

. . .

Authored by (in block letters):
For papers written by groups the names of all authors are required.

Name(s):	**First name(s):**
Muster	Student

With my signature I confirm that

- I have committed none of the forms of plagiarism described in the Citation etiquette information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work .
- I am aware that the work may be screened electronically for plagiarism.
- I have understood and followed the guidelines in the document *Scientific Works in Mathematics.*

Place, date:	**Signature(s):**
Zurich August 19th 2009	bla

For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.