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.

methods(not done)

results(not done)

Contents

List of Figures

List of Tables

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 their 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. 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.

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 convenient, parsimonious parametrization, that is of interest to study.

normal gives easy param ov cov mats multivariate builds on work done in the nor1mix package. 'filedrawer research'

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.

$\boldsymbol{Y}_1, \ldots, \boldsymbol{Y}_n$

Definition 1.1.0.1. *Suppose we have a random sample $\boldsymbol{Y}_1, \ldots, \boldsymbol{Y}_n$ with probability density function $\boldsymbol{Y}_j \sim f(y_j)$ on $\mathbb{R}^p$ We assume that the density $f(y_j)$ of $\boldsymbol{Y}_j$ can be written in the form*

$$f(y_j) = \sum_{k=1}^{K} \pi_k \phi_k(y_k; \mu, \Sigma)$$

The π_k are called the component densities of the mixture and the ϕ_k mixture components.

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, since it is the scope of this thesis and simplifies it considerably.

Suppose we have a $p-$dimensional dataset of n samples $x_1, \dots, x_n$, onto which we would like to fit K normal distributions $\phi_k, \;\; k \in 1, \dots, n$. We introduce a further explaining variable $\boldsymbol{Z}$ in $\mathrm{Mat}^{n \times k}$, with entries in $[0, 1]$ which represent the expectation that observation i belongs to component k.

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.

$$\tau_i(y_j; \Psi) = \phi_i(y_j; \mu_i, \Sigma_i) / \sum_{k=1}^{K} \phi_k(y_j; \mu_k, \Sigma_k)$$

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

$$\mu_i = \sum_{j=1}^{n} \tau_{ij} y_j / \sum_{j=1}^{n} \tau_{ij}$$

$$\Sigma_i = \sum_{j=1}^{n} \tau_{ij} (y_j - \mu_i)(y_j - \mu_i)^\top / \sum_{j=1}^{n} \tau_{ij}$$

here note about initialization methods.

While it is possible to use a purely EM-based approach, most popular implementations use some form of pre clustering and use the EM-algorithm as final pass to fit the data. The R-package `Mclust` for example uses hierarchical agglomerative clustering L, M, TB, and AE. (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

$$\Sigma = \lambda \boldsymbol{DAD}^\top$$

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 $\alpha \boldsymbol{LDL}^\top$ 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:

$$\begin{aligned} \boldsymbol{D} &\longmapsto \boldsymbol{Q} \\ \boldsymbol{A} &\longmapsto \boldsymbol{\Lambda} \\ \lambda &\longmapsto \alpha \\ \Sigma = \lambda \boldsymbol{DAD}^\top &= \alpha \boldsymbol{Q\Lambda Q}^\top \end{aligned}$$

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 eigenvectors and α was somewhat arbitrarily chosen.

1.4 Models of Covariance Matrices

make clear that the models can not be translated one to one to ldlt model 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:

$$\Sigma = \alpha \boldsymbol{Q\Lambda Q}^\top \quad \Sigma = \alpha \boldsymbol{LDL}^\top$$

Since in both cases the bracketing 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 & Grovaert 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 $\boldsymbol{LDL}^\top$ 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 $\boldsymbol{L}_k\boldsymbol{D}\boldsymbol{L}^\top$, 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	$\alpha \boldsymbol{Q\Lambda Q}^\top$	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	$\alpha \boldsymbol{Q\Lambda}_k \boldsymbol{Q}^\top$	equal	var.	equal	$\alpha, \lambda_{i,k}, q_{i,j}$	doesn't exist		$1 + K(p-1) + \frac{p(p-1)}{2}$
VEE	$\alpha_k \boldsymbol{Q\Lambda Q}^\top$	var.	equal	equal	$\alpha_k, \lambda_i, q_{i,j}$	$\alpha_k \boldsymbol{LDL}^\top$	$\lambda_k, d_i, l_{i,j}$	$K + p + \frac{p(p-1)}{2}$
VVE	$\alpha_k \boldsymbol{Q\Lambda}_k \boldsymbol{Q}^\top$	var.	var.	equal	$\alpha_k, \lambda_{i,k}, q_{i,j}$	don't exist		$K + K(p-1) + \frac{p(p-1)}{2}$
EEV	$\alpha \boldsymbol{Q}_k \boldsymbol{\Lambda Q}_k^\top$	equal	equal	var.	$\alpha, \lambda_i, q_{i,j,k}$			$1 + (p-1) + K\frac{p(p-1)}{2}$
VEV	$\alpha_k \boldsymbol{Q}_k \boldsymbol{\Lambda Q}_k^\top$	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.	$\alpha, \lambda_i, q_{i,j,k}$	$\alpha \boldsymbol{L}_k \boldsymbol{D}_k \boldsymbol{L}_k^\top$	$\lambda, d_{i,k}, l_{i,j,k}\ j > i$	$1 + pK + K\frac{p(p-1)}{2}$
VVV	$\alpha_k \boldsymbol{Q}_k \boldsymbol{\Lambda}_k \boldsymbol{Q}_k^\top$	var.	var.	var.	$\alpha_k, \lambda_i, q_{i,j,k}$	$\alpha_k \boldsymbol{L}_k \boldsymbol{D}_k \boldsymbol{L}_k^\top$	$\lambda_k, d_{i,k}, l_{i,j,k}\ j > i$	$K + pK + K\frac{p(p-1)}{2}$

Table 1.1: Table of Parameters

1.5 Problems of EM

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]. In common software implementations, 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.

```
>     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

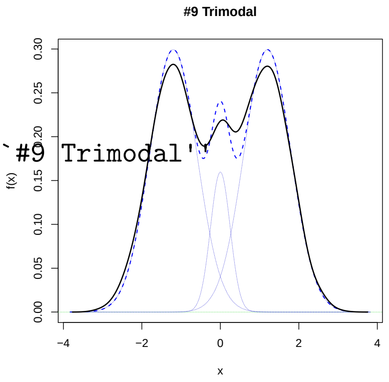


Figure 1.2: True and Estimated density

then an illustration of MW examples of pathological cases

here we see how change in loglik seems to stagnate. However, this does not stay that way, if we let EM run a bit further.

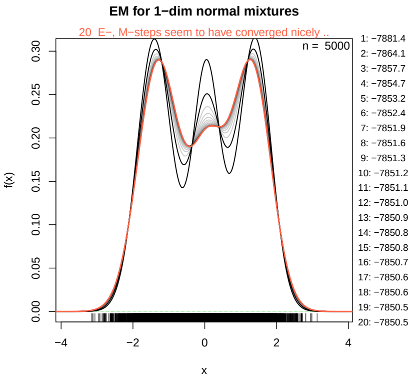


Figure 1.3: 20 EM steps

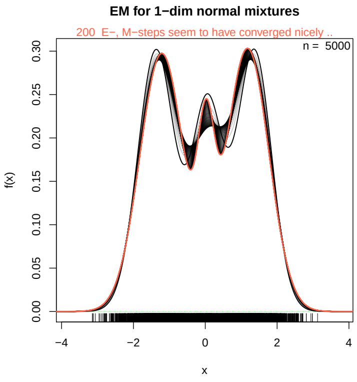


Figure 1.4: 200 EM steps

to conclude example show part of mixest that shows it takes 1200 iterations to converge

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.

maybe show Marr Wand’s examples of ’difficult’ mixtures

give conclusion recapping the just demonstrated, and lead in for next chapter

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. 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 Multivariate `mix`ture model, and the `llnorMmix()` function, that calculates the log-likelihood given a model and data.

The package contains the following functionality:

relies on `optim()` generic optimizer. maximizes llnormix by varying model parameters.

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.

Conceptually, at the start of a fitting algo, e.g. EM we need to initialize a mixture object. thereafter the paths diverge. at the heart of norMmix's functionality lie the functions: llnorMmix and nMm2par which are in turn employed by norMmixMLE to funnel a mixture object into optim and give optim a function to optimize.

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

In Notation	In Code
π_i	`w, weights`
Σ	`Sigma`
μ	`mu`
K	`k`
dimension	`p, dim, dims`
components	`cl, components`

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 13-th model of dimension 2.

simulations The purpose of this package is to study simulations. there are functions provided to study large collections of evaluated data. e.g `epfl`

2.2 On The Development of `norMmix`

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

maybe reread section in McLachlan about accelerating EM algo

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.

general list of (not necessarily mathematical) dead-ends in the development life of the norMmix package. argue why this is in this section?? because, as a BScT, the learning is as much part of the research as the results.

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] -3.1850438  0.7012354  0.6139481  0.5100746  0.5196708  0.6509962  0.6726699
[8]  0.5164488
```

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`:

```
> 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 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?

2.3 Demonstration

Mention, that mclust doesn't depend on seed(double check) and therefore norMmix has 'advantage' of 'confidence intervals'. We can run 50 simulations and see if there might be more sensible clusters.

demonstrate things; essentially put .Rd example sections here

Chapter 3

Comparing Algorithms

3.1 Time Analysis

here how much time they take, in p,k and n give approximate O(x) value

```
>     library(norMmix, lib.loc="~/ethz/BA/norMmix.Rcheck/")
>     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(log(times) ~ log(pars) + log(ddims) + log(ssize))
>     summary(r)

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

Residuals:
    Min      1Q  Median      3Q     Max 
-3.4428 -0.2986  0.0671  0.4579  2.0936 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept) -9.74133    0.10598  -91.91   <2e-16 ***
log(pars)    2.75983    0.01181  233.75   <2e-16 ***
log(ddims)  -2.06063    0.02483  -82.99   <2e-16 ***
log(ssize)   0.61301    0.01446   42.38   <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

```
Residual standard error: 0.6946 on 7196 degrees of freedom
Multiple R-squared:  0.8887,       Adjusted R-squared:  0.8887
F-statistic: 1.916e+04 on 3 and 7196 DF,  p-value: < 2.2e-16
```

```
>     plot(times~pars, log="xy", yaxt="n", xaxt="n")
>     sfsmisc::eaxis(1)
>     sfsmisc::eaxis(2)
```

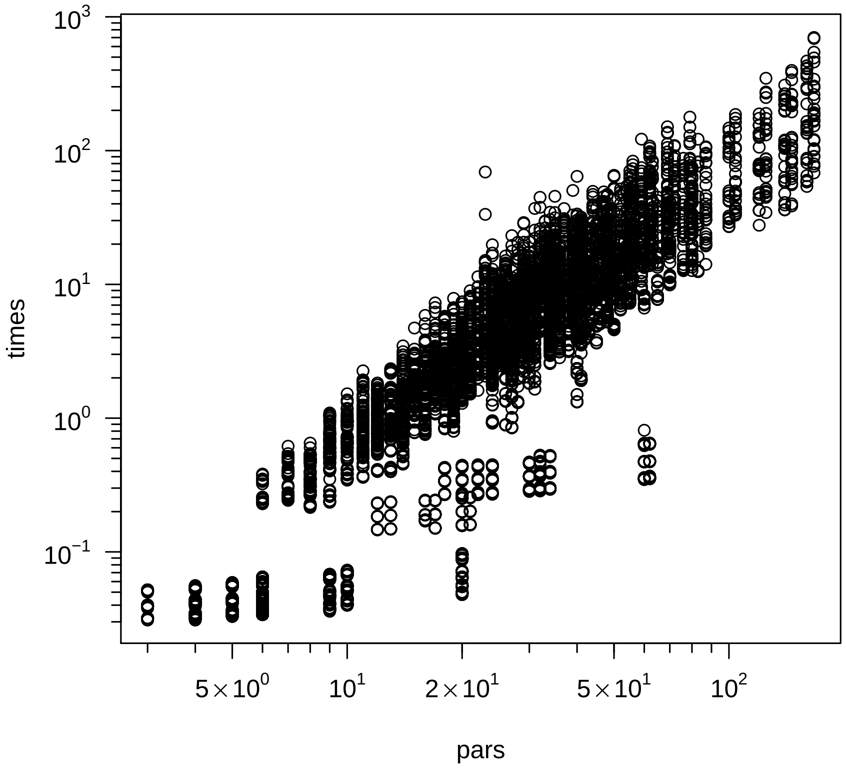


Figure 3.1: Log-log Plot of System Time against Parameter Length

can see that time is almost one to one proportional to parameter length.

3.2 Behaviour in n

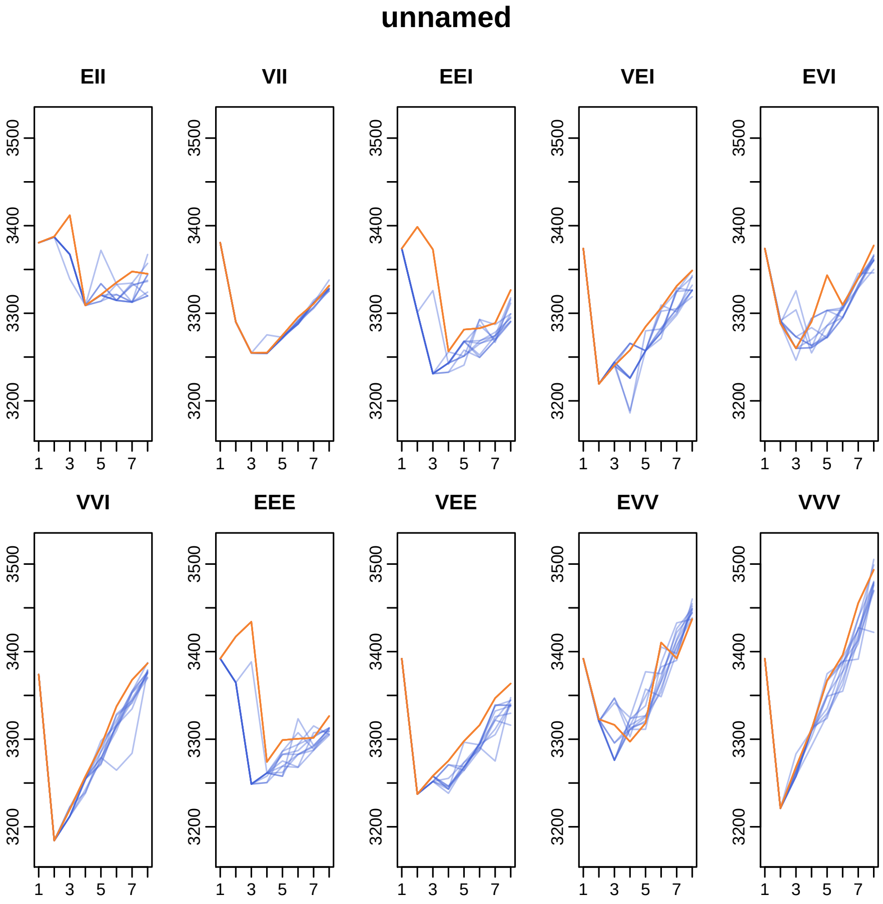
here show as expected narrower scattering as n increases

[h]

3.3 Behaviour in p

here show how norMmix is consistently competitive with mclust

```
>     compplot(s05mw34bic, m0534)
```



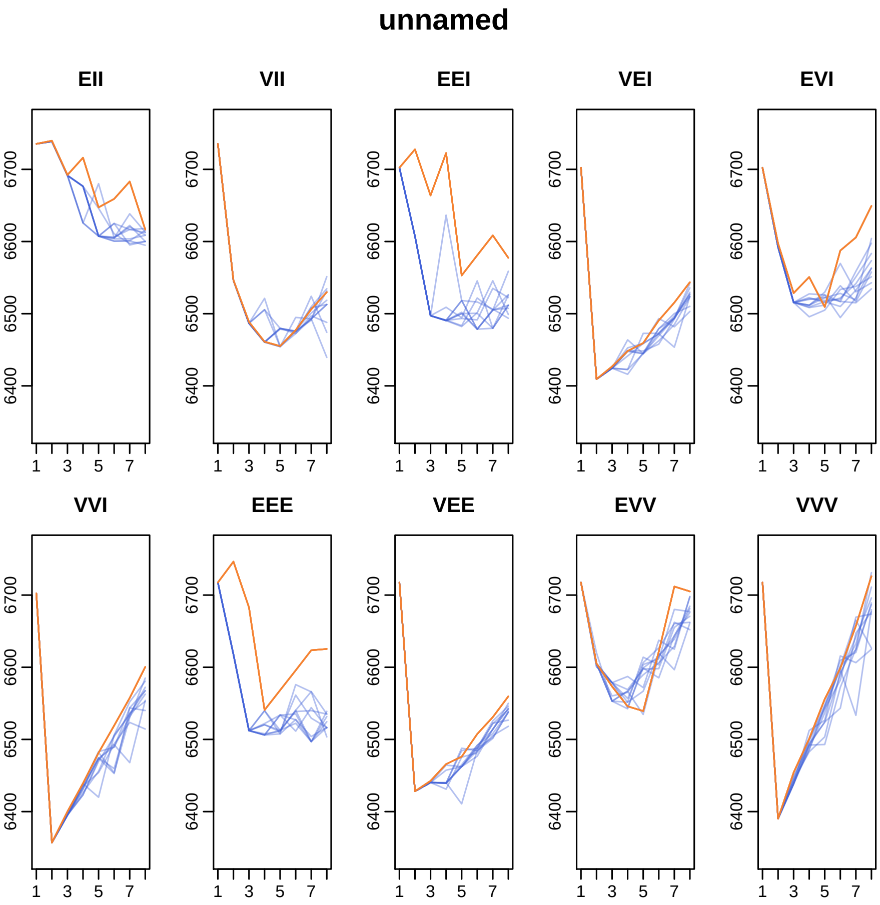
3.4 Diffixult Mixtures

here show behaviour in difficult cases

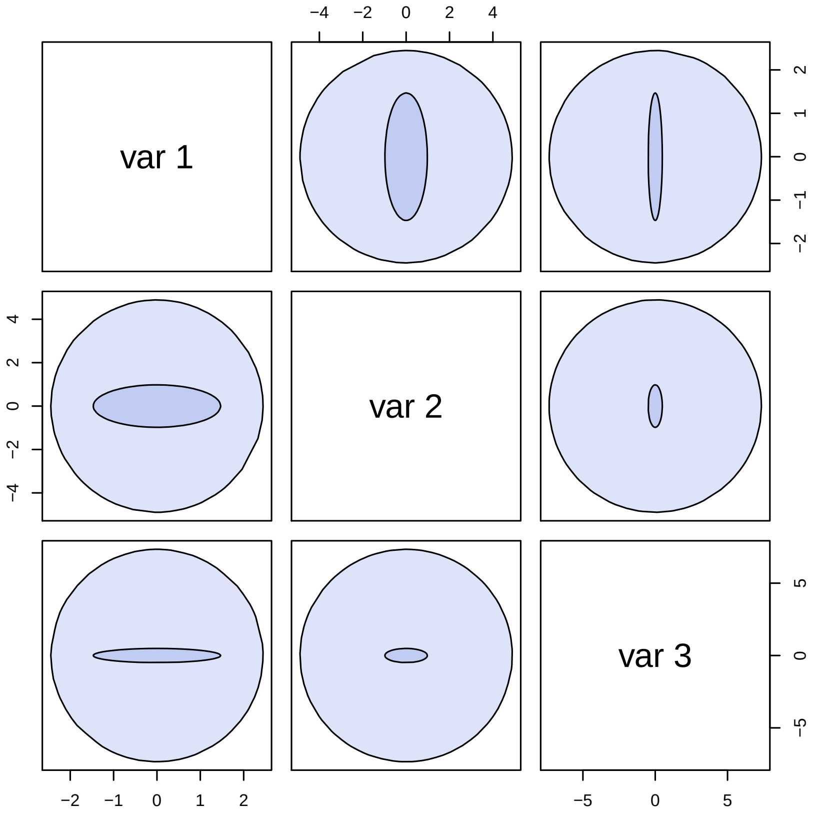
```
>     savdir <- file.path(mainsav, "2init")
>     filenames <- list.files(savdir, pattern=".rds")
>     MW214fn <- grep("MW214", filenames, value="TRUE")
>     mclustfiles <- grep("mcl.rds", MW214fn, value=TRUE)
>     MW214fn <- grep("mcl.rds", MW214fn, value="TRUE", invert=TRUE)
>     claraMW <- grep("clara", MW214fn, value=TRUE)
>     mclMW <- grep("mclVVV", MW214fn, value=TRUE)
>     clarabic <- massbic(claraMW, savdir)
>     mclbic <- massbic(mclMW, savdir)
>     mclustbic <- readRDS(file.path(savdir,mclustfiles[1]))
```

3.5 Nonnormal Mixtures

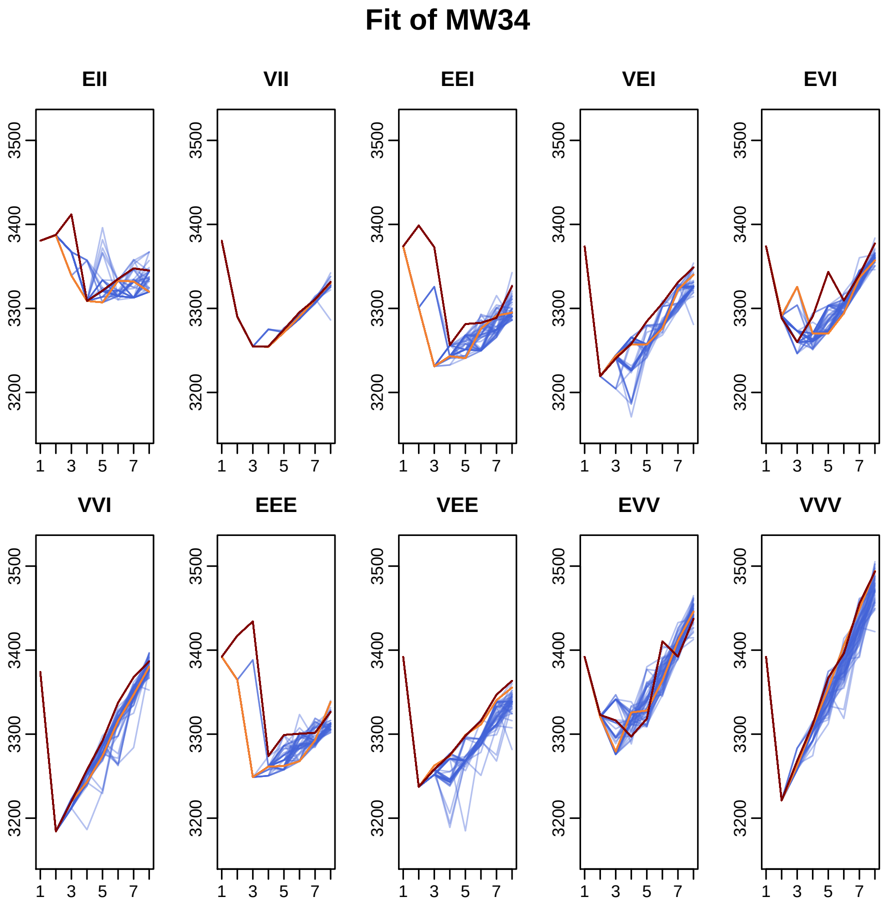
```
>       compplot(s10mw34bic, m1034)
```



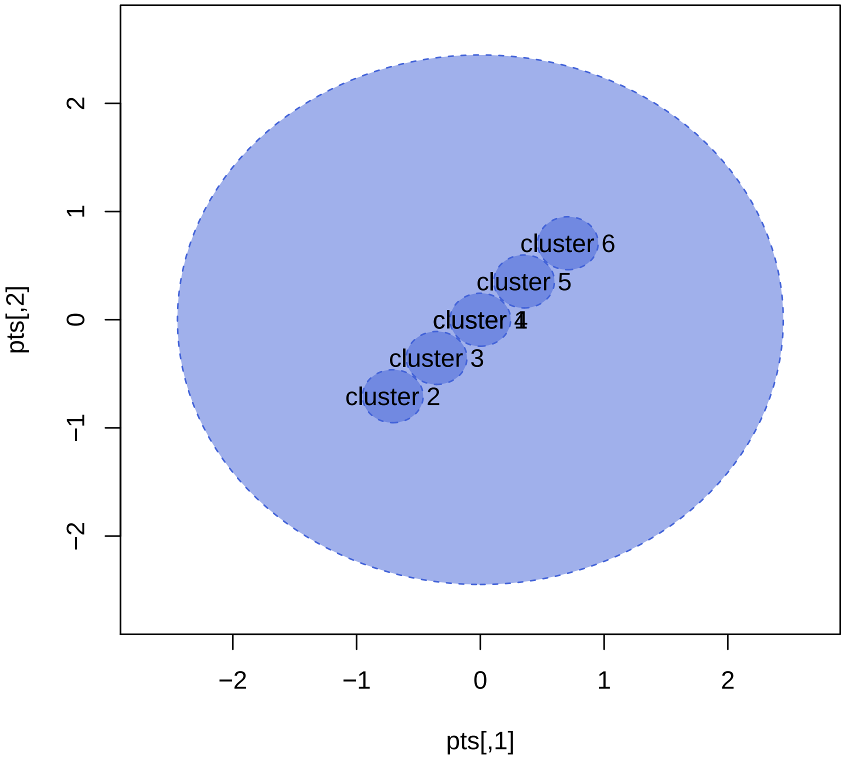
```
>     plot(MW34)
```



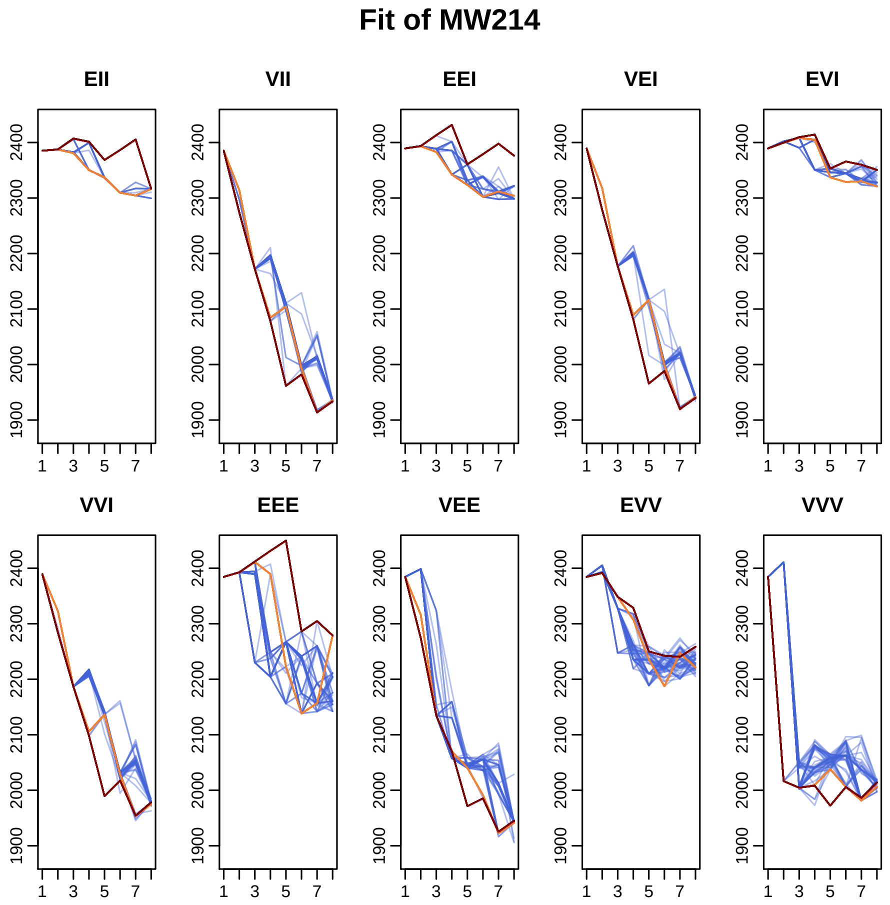
```
>     compplot(clarabic, mclbic, mclustbic, main="Fit of MW34")
```



```
>       plot(MW214)
```


cluster 6
cluster 5
cluster 1
cluster 4
cluster 3
cluster 2
−2
−1
0
1
2
pts[,1]
pts[,2]

```
>     compplot(clarabic, mclbic, mclustbic, main="Fit of MW214")
```



Chapter 4

Discussion

one shortcoming is time inefficiency. largely due to implementation. mclust has 16'000 lines of Fortran code, impossible in the scope of this thesis.

proof of concept?? definitely possible to do model selection using a general optimizer.

strong points: 'randomness' of clara/optim allows 'confidence intervalls' for selected model flexibility of approach: given an logLik fctn can do mixture fitting w/ arbitrary models

further study might include: other presumed component distributions, 'high' dimensions

Bibliography

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

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

L, S., F. M, M. TB, and R. AE. (2016). mclust 5: Clustering, classification and density estimation using gaussian finite mixture models. *R J.* (8(1)), 289–317.

Marron, J. S. and M. P. 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.&#x2014;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.

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.[,
            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))
}
sllnorMmix <- function(x, obj, trafo=c("clr1", "logit")) {
    stopifnot(is.character(model <- obj$model))
    trafo <- match.arg(trafo)
    llnorMmix(nMm2par(obj, model=model),
              tx = t(x), k = obj$k,
              model=model, trafo=trafo)
}
## log-likelihood function relying on mvtnorm function
#
# par:   parameter vector as calculated by nMm2par
# x:     matrix of samples
# k:     number of cluster
# trafo: transformation of weights
# model: assumed model of the distribution
llmvtnorm <- function(par, x, k,
                      trafo=c("clr1", "logit"),
                      model=c("EII","VII","EEI","VEI","EVI",
                              "VVI","EEE","VEE","EVV","VVV")
              ) {
    stopifnot(is.matrix(x),
              length(k <- as.integer(k)) == 1, k >= 1)
    model <- match.arg(model)
    trafo <- match.arg(trafo)
    p <- ncol(x)
    nmm <- par2nMm(par, p, k, model=model, trafo=trafo)
    ## FIXME (speed!):  dmvnorm(*, sigma= S) will do a chol(S) for each component
    ## -----  *instead* we already have LDL' and  chol(S) = sqrt(D) L' !!
    ## another par2*() function should give L and D, or from that chol(Sagma), rather than S
    w <- nmm$w
    mu <- nmm$mu
    sig <- nmm$Sigma
    y <- 0
    for (i in 1:k) {
        y <- y + w[i]*mvtnorm::dmvnorm(x,mean=mu[,i],sigma=sig[,,i])
    }
    sum(log(y))
}
```

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)
```

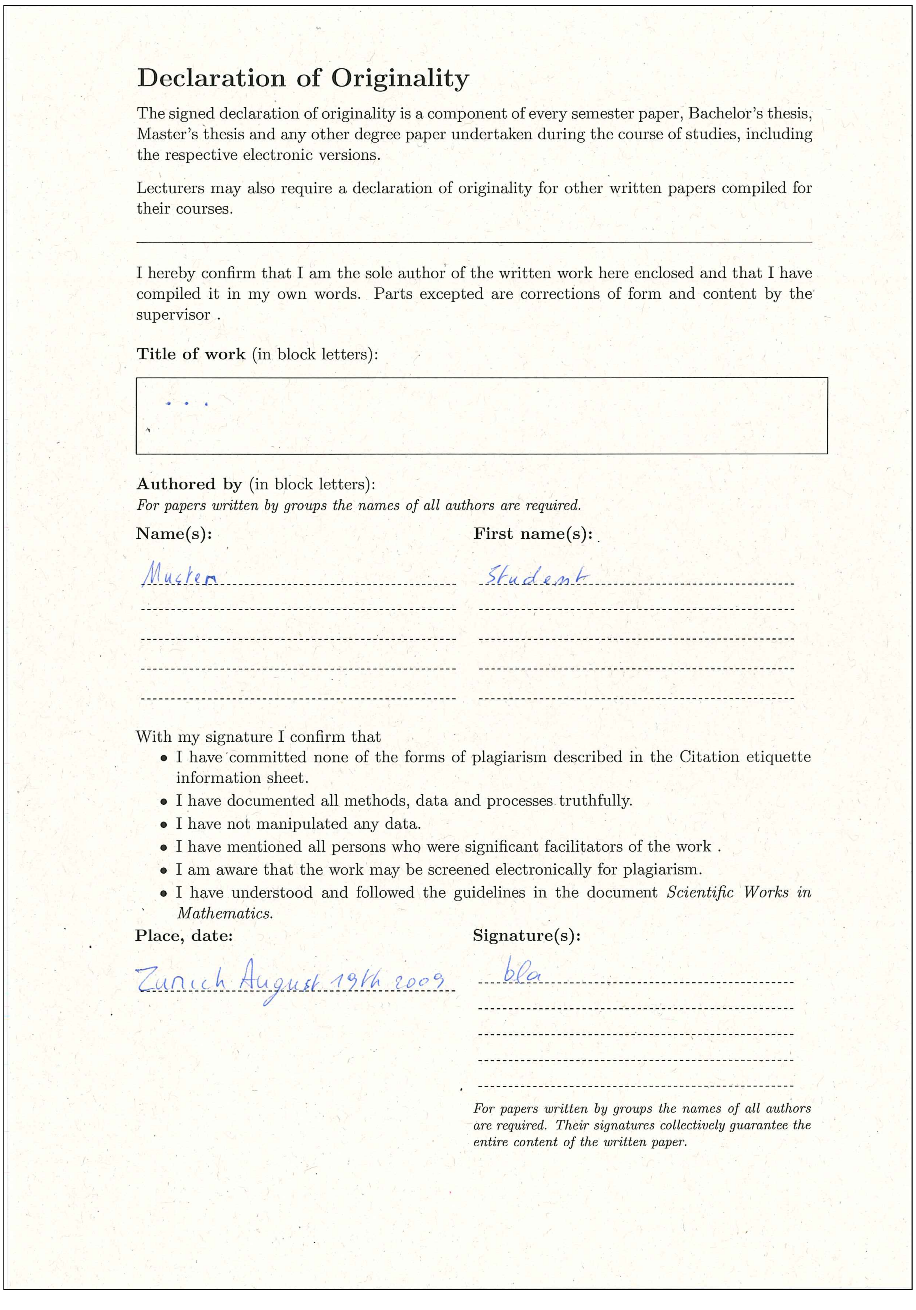
Appendix B

Further Plots

here further plots:

B.1 Ch3

dsfasdf



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): Muster

First name(s): 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: Zurich August 19th 2009

Signature(s): bla

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