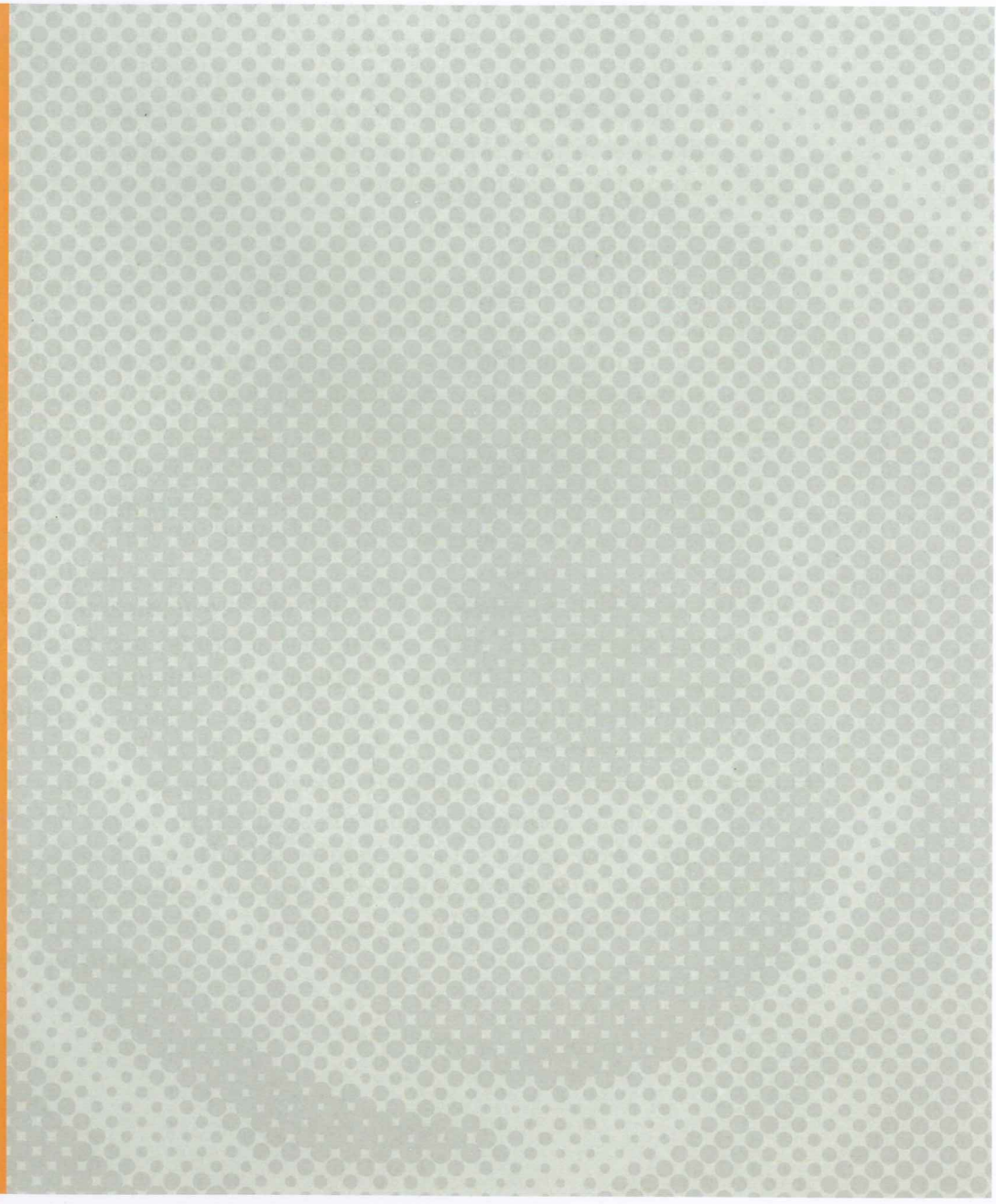
Estimation of mixture-distributions using the Direct Look-up method

Estimering av blandede fordelinger med Direct Look-Up metoden

Kristine Navestad

Norwegian University of Life Sciences
Department of IKBM
Master Thesis 60 credits 2012



Abstract

Mixture distributions and models are useful methods of describing data that cannot be estimated with a single probability distribution. Estimating mixture models based on samples from unknown distributions is a highly iterative process, prone to issues like non-convergence, high runtime or local optima. It is therefore an interesting area to develop a method of parametric estimation without some of these issues using a direct approach without iteration. Isaeva et al. [2011a,b] developed a method to avoid these problems regarding estimation of non-linear mathematical functions called the Direct Look-Up (DLU) method. To implement this method for estimating mixture models is the main goal of this master thesis.

The idea is to compute a wide range of possible solutions to the parametric estimation problem of mixture distributions prior to observing data. Within a standardized range, a set of possible combinations of two normal distributions are chosen according to an experimental design, and the mixture distributions are computed. Using Principal Component Analysis (PCA) the generated library of distribution curves is reduced to a number of basal curves, storing vast amount of information into a few vectors and values. Finding the best solutions to a new observation is then a matter of linear projection onto the curve library to identify the nearest curves. The method is to be compared to traditional estimation methods for mixture models such as Expectation Maximization (EM) and Markov Chain Monte Carlo (MCMC). Results are evaluated and compared using log likelihood values.

The general performance of the DLU for mixture models is acceptable. Many observations can be estimated quickly, with a predictable time frame. Results in comparison with other methods show that it is not as optimized as the EM for this problem, while it works better than MCMC. Differences in prediction error and log likelihood evaluations of the estimated parameters are discussed and analysed. Possibilities for taking advantage of the generalizable nature of the DLU method is discussed. Using the DLU for mixture models is a method that may at some point work well, even though the method needs some tweaks to make it better. Generalising the method for a larger collection of mixture models is discussed, and ideas for improvement are presented.

Sammendrag

Blandete sannsynlighetsfordelinger beskriver data som ikke lett kan beskrives i en enkel sannsynlighetsfordeling. Estimering av blandete sannsynlighetsmodeller basert på tilfeldige observasjoner fra ukjente fordelinger er en iterativ prosess, som er utsatt for problemer som inkonvergens, lang estimeringstid eller å kun finne lokale optimum. Dette er derfor et spennende område for å utvikle nye metoder. Ved hjelp av en direkte framgangsmåte uten iterasjon kan vi lage en parametrisk estimeringsmetode der noen av disse problemene kan unngås. I Isaeva et al. [2011a,b] ble en metode utviklet for å unngå disse problemene i forhold til estimering av ikke lineære matematiske funksjoner kalt "Direct Look-Up" (DLU). Å implementere denne metoden for blandete sannsynlighetsmodeller er hovedmålet i denne masteroppgaven.

Den generelle ideen er å beregne en rekke mulige løsninger til estimeringsproblemet på forhånd. Innenfor standardiserte områder er et sett av mulige kombinasjoner av to normalfordelinger valgt ut etter forsøksdesign og den blandete sannsynlighetsfordelingen er beregnet for disse. For å kunne lagre store mengder data i noen få vektorer og verdier, anvendes prinsipal komponent analyse (PCA) på det genererte biblioteket av fordelingskurver. Det å finne de beste løsningene blir da å projisere nye observasjoner på kurvebiblioteket for å finne de nærmeste kurvene. Metoden sammenlignes med tradisjonelle metoder som Expectation Maximization (forventnings maksimisering, EM) og Markov Chain Monte Carlo (MCMC) for å evaluere resultatene. Resultatene er evaluert og sammenlignet ved hjelp av log verdiene av sannsynlighetsfunksjonen, "log likelihood".

Den generelle prestasjonsevnen til DLU for blandingsmodeller er akseptabel. Mange observasjoner kan estimeres raskt, innenfor en tidsramme som er kjent på forhånd. Resultater i sammenligning med andre metoder viser at DLU er ikke like optimalisert for blandingsmodeller som EM, mens den yter bedre resultater enn MCMC. Forskjeller i prediksjonsfeil og log likelihood verdier av de estimerte parametrene blir diskutert og analysert. Det å bruke DLU for blandete sannsynslighetsmodeller er en metode som vil kunne fungere bra, selv om metoden slik den er nå trenger noen endringer for å gjøres bedre. Det er diskutert å generalisere metoden til en større samling av blandete sannsynlighetsfordelinger, og ideer for forbedring er presentert.

Acknowledgements

This thesis is written at IKBM, UMB, as part of the Biostatistics group, supervised by Associate Professors Solve Sæbø and Trygve Almøy.

Method development has been a process that is relatively new to me, the trying and testing and discussion of what and how and why, yet still tremendously exciting, doing something a bit different from what has been done before. I loved the new idea presented from the moment I heard of it, and wanted to take part in the development of the process. Everyone wishes to change the world with their work, and some reality checks are called for when it comes to what can be achieved and tested during a master thesis, but it has been an exciting process.

Working with this project over this last year has been an educational process, and I would like to thank all the wonderful people who have helped me through it; my supervisors Solve Sæbø and Trygve Almøy, whose doors are always open, no matter what the next question might be, all my friends and family who always thinks I will do great, for listening to me trying to make sense of it all and pretending to understand what I am talking about and reading what I write. Thanks to my dear Remi, who has supported me and help keep me sane in all of this.

Happy reading,

Kristine Navestad

Ås, May 2012

Contents

List of Tables

List of Figures

Chapter 1

Notes on terminology and notation

1.1 Notation

During this thesis a common set of symbols will be used to explain different elements and matrix computation. A set of basic terms will also be explained here to avoid confusion when continuing.

Stochastic variables is something that can be measured or gathered information about, an event that takes one of several possible outcomes. This can for example be weight of an animal, the presence of a specific bacteria in a growth, etc. Variables are denoted with latin letters, such as a, x and X.

A parameter is an unknown number describing a property of the random variable. Examples are the expected weight of an animal, the true probability of an event happening or not, or the true variation in the population. Parameters are denoted in Greek letters, such as α, β and γ. μ is commonly used as a parameter for the expected value in a population, while σ^2 is used for the population variance. Symbols used for one parameter describing something here might mean something different in another subject, the use of symbols for parameters is not mutually exclusive.

Common properties of variables include the expectation μ and the variance

σ^2. The expected value of a random variable X is denoted as

$$\mu = E(X)$$

and the variance of the variable is denoted as

$$\sigma^2 = Var(X)$$

A random vector $\mathbf{X} = (X_1, \ldots, X_k)^T$ is a collection of k random variables. A random variable can be generalized as a random vector with $k = 1$. For more specifics about vectors and matrices, see for instance Bickel and Doksum [2007,1977, A.8].

Several random vectors can be combined to a matrix, which is a collection or row vectors and column vectors such as

$$X_{k,n} = \begin{pmatrix} X_{1,1} & X_{1,2} & \cdots & X_{1,n} \\ X_{2,1} & X_{2,2} & \cdots & X_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ X_{k,1} & X_{k,2} & \cdots & X_{k,n} \end{pmatrix}$$

Vectors can be generalized as a $k \times 1$ matrix. Matrix computation has more advanced rules than computation with just numbers, and they can all be found in more detail in for example either Lay [2006], Johnson and Wichern [2007] or Bickel and Doksum [2007,1977], or any other textbook in statistics or linear algebra.

Chapter 2

Introduction

2.1 Problem

The goal of statistical modelling is to achieve insight over what happens around us. Mathematical models can be used to explain many things in the world and are used to achieve some sense of control over what happens in the physical world. The movement of falling objects, population growth, physical phenomena, the list goes on. Many of these models will present difficulties fitting a model to the data. In the case of non-linear models or mixture models, parametric model estimation can be complicated. Mixture distribution can be estimated by a variety of methods, but they are in general iterative and computational heavy. The final results will depend on the input parameters of where to start the computation, convergence issues that might appear, leaving us without a solution at all, or it will simply take more computing time than possible. The iterative process can also find a local optimum, and from that point insist that there are no better solution, even though it is not the final optimum solution to the situation. These are problems that may create issues when using traditional iterative methods.

The development of new, faster methods is therefore important to save time and effort. The Direct Look Up method presented in Isaeva et al. [2011a,b] is a new method for estimating non linear functions. This method, from this point referred to as DLU, for estimating curves is a fast new way of estimating complicated functional relations. It uses a pre-built curve library for fast estimation of new curves through projection of the new curves on to the compressed database of curves. Its focus is monotonous curves with one inflection point.

The genius of this method is that it needs no previously known assumption regarding the data. A wide range of possible solutions are

tested at the same time, it is not necessary to pick a few distinct mathematical functions to run. This allows the user to find solutions he might not otherwise have thought of. To take this further, one might wish to look at other functions as well, not just monotonous functions with one inflection point as was used in the development of the original DLU method, but also mixture distributions and functions. When looking at combinations of different distributions, or multiple versions of one distribution, the variations are endless.

Mixture distributions and models often occur in real life situations, and it is interesting to identify the distribution components of such mixtures. Implementing the DLU method for parametric estimation of mixture distributions is an effort to use the non iterative ideas of the DLU for a new area of model estimation. As a start, the DLU method will be implemented for a mixture of two normal distributions, relevant data will be simulated and the method applied. Comparisons with other computational methods will be performed, looking both at runtime and statistical error. This will result in whether or not to recommend this method for further work, implementation and use, and whether or not extending the database will be interesting.

Before the development of the method, an introduction to the theory needed during the development is presented.

2.2 Probability distributions

The method to develop is a method for parametric estimation of mixture distributions. To find a parametric estimate of this distribution a look at what probability distributions are is in order. A probability distribution describes the probability of the outcome of a variable. They are expressed as either a probability density function or a probability distribution function.

There is a set of attributes specifying a function $f(x)$ as a probability density function for a random variable x.

1. The total area under the function curve is always one.

$$\int_{-\infty}^{\infty} f(x)dx = 1$$

2. The function is always non-negative.

$$f(x) \geq 0$$

3. The probability of an outcome between the numbers a and b equals the integral of $f(x)$ from a to b.

$$P(a < x < b) = \int_a^b f(x)dx$$

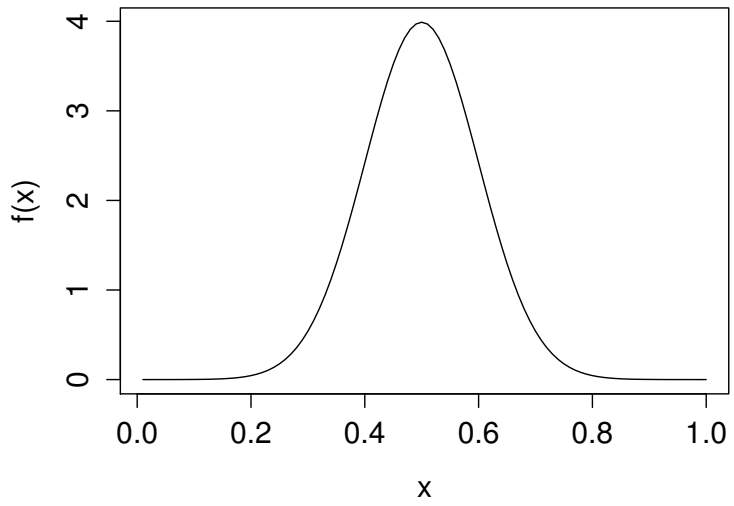


Figure 2.1: An example of a probability density function, the Normal distribution

A probability distribution function is recognized by having a range between 0 and 1 and the following properties:

1. The minimum value is 0 and the maximum value is 1

$$\lim_{x \to -\infty} F(x) = 0$$
$$\lim_{x \to \infty} F(x) = 1$$

2. F is an increasing function,

$$F(a) > F(b) \quad \text{for all} \quad a > b$$

It follows that the probability of an outcome between a and b is expressed by

$$P(a < x < b) = \int_a^b f(x)dx = F(b) - F(a)$$

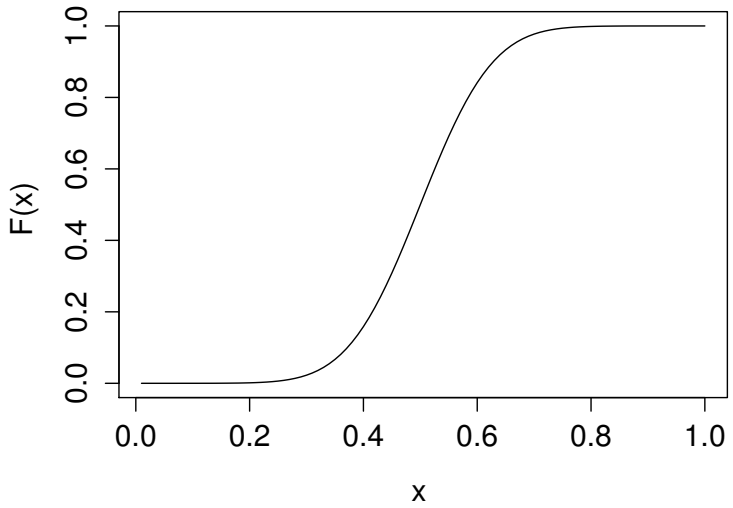


Figure 2.2: An example of a probability distribution function, the Normal distribution

Estimation of the functions $f(x)$ and $F(x)$ and the corresponding parameters from observed data are essential to statistics. Graphical illustration of the density function $f(x)$ and the distribution function $F(x)$ are seen in the figures 2.1 and 2.2 respectively.

Examples of parametric probability distributions include the Normal distribution, Binomial distribution, Poisson, Fischer, t and chi-square. These functions are used because so many natural phenomena tend to be well described by one of these under certain circumstances, and are very useful in determining probability outcomes.

The distributions are used in notation as

$$X \sim D(\eta)$$

This notation is used to express that a stochastic variable X follows a distribution D with a set of parameters η. There are many situations that can be transformed into following a particular distribution. This knowledge is used to set up a statistical model for the problem.

The most commonly used distribution is the Normal Distribution (or the

Gaussian distribution), following the probability function:

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{2.1}$$

with the parameters μ and σ. The parameter μ is the expected value of x, $\mu = E(x)$ and σ is the standard deviation of x, $\sigma = \sqrt{Var(x)}$.

The Central Limit theorem states that the sum of a large number of random variables tends towards a normal distribution. This is of great importance, and is the basis for much asymptotic statistical theory. The theorem is found in e.g. Bickel and Doksum [2007,1977, p.470-471]

2.3 Mixture distributions and models

A mixture model is a model based upon an assumption of a mixture of distributions, displaying the probability distribution of a population with unknown sub-populations.

Take the heights of individuals in a selection as an example. We only know the heights, we don't have any of the other information. A probability distribution will then reflect the different sub-populations that we might not know about. A mixture model is a model explaining the unknown sub-populations.

In general, a mixture model over the points

$$\{x_1 \dots x_n\}$$

can be stated as a g-component mixture probability density function:

$$f(x_i; \boldsymbol{\theta}) = \sum_{j=1}^{g} \varepsilon_j f_j(x_i; \boldsymbol{\theta}_j) \tag{2.2}$$

and can take the visual form as illustrated in the example figure 2.3

The mixing proportions, ε_j, are non negative and sum to one. The function $f(x_i; \boldsymbol{\theta})$ is the probability density function given the parameter set $\boldsymbol{\theta}$, as a weighted sum of density functions following parameter sets $\boldsymbol{\theta}_j$.

A mixture model of only two components (g=2) is stated as:

$$f(x; \boldsymbol{\theta}) = \varepsilon f_1(x; \boldsymbol{\theta}_1) + (1 - \varepsilon) f_2(x; \boldsymbol{\theta}_2) \tag{2.3}$$

The number of distributions is not limited to just two at a time, it can be multiple distributions. The individual distributions can be from different families of distributions, but usually distributions are from the same family.

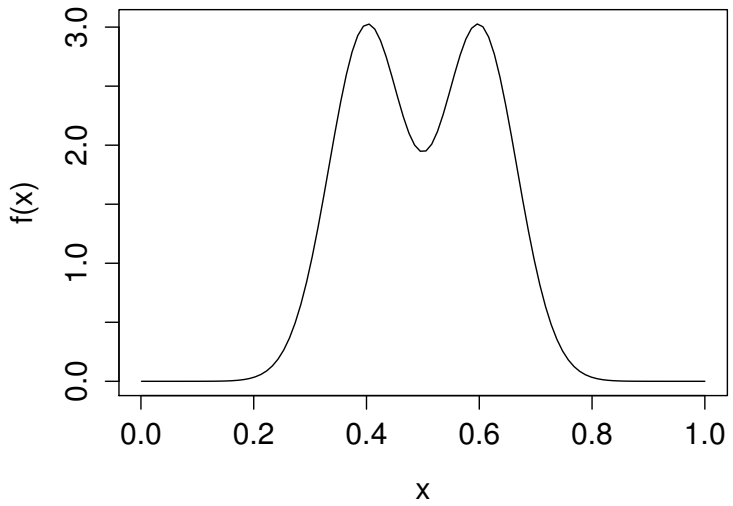


Figure 2.3: An example of a mixture of normal distributions with the parameters $\varepsilon = 0.5$, $\mu_1 = 0.6$, $\mu_2 = 0.4$, $\sigma_1 = 0.2/3$ and $\sigma_2 = 0.2/3$

The mixture model can be used in the case of one ordinary density function not explaining all variation in the data. This can for instance be when there are unknown sub-populations in the data. Mixture modelling allows these sub groups to be identified.

As for other distribution functions, a mixture distribution also follows the properties for probability distributions, such as $\int_{-\infty}^{\infty} f(x)dx = 1$ and that $F(0) = 0$ and $F(1) = 1$.

The number of parameters in a mixture model depends on the distributions the mixture is made up of. Parameters in the different components of the mixture are independent of each other, and the final number of parameters will be the total number of parameters in the separate parts, in addition to mixing proportions. If there are g components in the mixture, then there will be $g - 1$ parameters for the mixing proportions to estimate since the g proportions sum to one, $\sum_{j=1}^{g} \varepsilon_j = 1$.

2.4 Estimation methods

To estimate parameters for any model, estimators are used. An estimator is a function of data that give estimates of an unknown parameter. Common

estimators include e.g. the sample mean as an estimator for the unknown expectation of the population and the sample standard deviation as a measure of the unknown deviation in the population. There are many classes of estimators. They are evaluated based on different criteria, such as unbiasedness, minimum variation and maximum likelihood.

There are several different approaches to parameter estimation of the mixture model that are already established methods. Two different methods are presented here: Expectation Maximization (EM) and Markov Chain Monte Carlo (MCMC). Both are established methods for estimating mixtures of normals, but in different ways.

The EM and MCMC algorithms work on the likelihood function of the mixture. The estimation problem can be defined as a mixture of two normals such as in section 2.3, expressed as:

$$f(x) = \varepsilon N(x; \mu_1, \sigma_1^2) + (1 - \varepsilon)N(x; \mu_2, \sigma_2^2) \tag{2.4}$$

A very simplified way to look at this is:

$$f(x) = \varepsilon_1 f_1(x, \boldsymbol{\theta}) + \varepsilon_2 f_2(x, \boldsymbol{\theta})$$

Let the vector $\mathbf{z}$ contain the values $\{1, 2\}$ defining whether an element in $\mathbf{x}$ is from $f_1(x)$ or $f_2(x)$. The set of all parameters involved is $\boldsymbol{\theta} = \{\mu_1, \mu_2, \sigma_1^2, \sigma_2^2, \varepsilon_1, \varepsilon_2\}$. If $\mathbf{z}$ was known, we would have the simultaneous probability distribution of x and z, given $\boldsymbol{\theta}$.

$$f(\mathbf{x}, \mathbf{z}|\boldsymbol{\theta})$$

By using Bayes theorem, as stated in any introductory statistics book this can be split to

$$f(\mathbf{x}, \mathbf{z}|\boldsymbol{\theta}) = f(\mathbf{x}|\mathbf{z}, \boldsymbol{\theta}) f(\mathbf{z}|\boldsymbol{\theta})$$

The likelihood function for a maximum likelihood estimator for $\boldsymbol{\theta}$ would be:

$$[\prod_{j=1}^{2} \prod_{i \in I_j} f(x_i|z_i, \mu_j, \sigma_j^2)] \prod_{i=1}^{n} f(z_i|\boldsymbol{\theta}) \tag{2.5}$$

as shown by Lopes [2005].

This defines the background for estimation using traditional methods for mixture model estimation, such as MCMC and EM.

2.4.1 Expectation Maximization

The EM algorithm for Maximum Likelihood Inference is used to estimate model parameters from a model as described in section 2.4. The density function, or likelihood function, to be estimated is the same as previously.

$$f(\mathbf{x}, \mathbf{z}|\boldsymbol{\theta}) = l(\boldsymbol{\theta}|\mathbf{x}, \mathbf{z}) \tag{2.6}$$

If it was known an ML estimate would maximize the log likelihood function. (Lopes [2005])

$$\boldsymbol{\theta} = \arg\max_{\theta} l(\boldsymbol{\theta}|x, z)$$
$$\boldsymbol{\theta} = \arg\max_{\theta} log(f(\mathbf{x}, \mathbf{z}|\boldsymbol{\theta}))$$

However, since equation 2.4 is unknown we rather maximize the expected log likelihood over the distribution $f(z\|\boldsymbol{\theta})$. The algorithm is as follows:

Initial values for $\boldsymbol{\theta} = \boldsymbol{\theta}^0$ and a distribution for $f(\mathbf{z}|\boldsymbol{\theta})$ are chosen to be able to perform the algorithm.

Computing the expected log likelihood function defines the E-step of the algorithm;

$$Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{(l)}) = \int log(f(\mathbf{x}, \mathbf{z}|\boldsymbol{\theta}) f(\mathbf{z}|\boldsymbol{\theta}))d\mathbf{z} \tag{2.7}$$

The M-step of the algorithm is updating the parameters after the current maximum likelihood estimate;

$$\hat{\boldsymbol{\theta}}^{(l+1)} = \arg\max_{\theta} Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{(l)}) \tag{2.8}$$

The algorithm iterates over these two steps until there are convergence in the parameter values.

2.4.2 Markov Chain Monte Carlo

From the definition above, $\boldsymbol{\theta}$ is the set of unknown parameters to estimate. Let $\mathbf{D}$ be the data that is observed.

$f(\mathbf{D}|\boldsymbol{\theta})$ is the likelihood function for $\boldsymbol{\theta}$. $\hat{\boldsymbol{\theta}} = \arg\max_{\theta} f(\mathbf{D}|\boldsymbol{\theta})$ is the Maximum Likelihood estimator. MCMC is a Bayesian inference algorithm. In Bayesian statistics it is assumed that $\boldsymbol{\theta}$ has a prior probability distribution, $f(\boldsymbol{\theta})$. This prior distribution is used to find the posterior distribution of $\boldsymbol{\theta}$.

The posterior distribution is found using Bayes theorem;

$$f(\boldsymbol{\theta}|\mathbf{D}) = \frac{f(\mathbf{D}|\boldsymbol{\theta})f(\boldsymbol{\theta})}{f(\mathbf{D})} \tag{2.9}$$

This is a Bayes estimate for $\boldsymbol{\theta}$. The Bayes estimator for $\boldsymbol{\theta}$ becomes:

$$\hat{\boldsymbol{\theta}}_B = \int \boldsymbol{\theta} f(\boldsymbol{\theta}|\mathbf{D}) d\boldsymbol{\theta} \tag{2.10}$$

where the posterior distribution of $\boldsymbol{\theta}$ is unknown and must be estimated. What is known about this distribution is that it is proportional to known distributions.

$$f(\boldsymbol{\theta}|\mathbf{D}) \propto f(\mathbf{D}|\boldsymbol{\theta})f(\boldsymbol{\theta})$$

MCMC is used to estimate the posterior distribution, $f(\hat{\boldsymbol{\theta}}|\mathbf{D})$, by sampling values $\boldsymbol{\theta}_i$ from $f(\boldsymbol{\theta}|\mathbf{D})$, $i = 1 \dots B$. The Bayes estimator is then found by:

$$\hat{\boldsymbol{\theta}}_B = \frac{1}{B}\sum_{i=1}^{B} \boldsymbol{\theta}_i \tag{2.11}$$

which is a Monte Carlo result.

The posterior distribution is proportional to that of the prior distribution and data given the parameters. For the case of a mixture of two normal distributions, $\boldsymbol{\theta}$ will include the parameters:

$$\boldsymbol{\theta} = \{\mu_1, \mu_2, \sigma_1^2, \sigma_2^2, \varepsilon_1, \varepsilon_2\}$$

The data D are $\{\mathbf{x}, \mathbf{z}\}$. The problem is that $\mathbf{z}$ is unknown. As shown earlier, it is known that $f(\mathbf{x}, \mathbf{z}|\boldsymbol{\theta}) = f(\mathbf{x}|\mathbf{z}, \boldsymbol{\theta})f(\mathbf{z}|\boldsymbol{\theta})$. Hence, the posterior distribution is:

$$f(\boldsymbol{\theta}|\mathbf{x}, \mathbf{z}) \propto f(\mathbf{x}|\mathbf{z}, \boldsymbol{\theta})f(\mathbf{z}|\boldsymbol{\theta})f(\boldsymbol{\theta})$$

where $f(\mathbf{x}|\mathbf{z}, \boldsymbol{\theta})$ is known is $\mathbf{z}$ is known and $\mathbf{z}$ is sampled from $f(\mathbf{z}|\boldsymbol{\theta})$ to get "full data".

Usually, the prior probabilities will be independent and the prior distribution

$$f(\boldsymbol{\theta}) = f(\mu_1)f(\mu_2)f(\sigma_1^2)f(\sigma_2^2)f(\varepsilon_1)f(\varepsilon_2)$$

The distribution of the single parameters are known to be proportional $f(\mathbf{D}|\boldsymbol{\theta})f(\boldsymbol{\theta})$ as well as for all the parameters. Values $z_1 \dots z_n$ are sampled

form $f(z|\boldsymbol{\theta})$. The conditional distributions are:

$$
\begin{aligned}
f(\mu_1|\mathbf{D},\mu_2,\sigma_1^2,\sigma_2^2,\varepsilon_1,\varepsilon_2) &\propto f(\mathbf{D}|\mu_2,\sigma_1^2,\sigma_2^2,\varepsilon_1,\varepsilon_2)f(\mu_2,\sigma_1^2,\sigma_2^2,\varepsilon_1,\varepsilon_2)\\
f(\mu_2|\mathbf{D},\mu_1,\sigma_1^2,\sigma_2^2,\varepsilon_1,\varepsilon_2) &\propto f(\mathbf{D}|\mu_1,\sigma_1^2,\sigma_2^2,\varepsilon_1,\varepsilon_2)f(\mu_1,\sigma_1^2,\sigma_2^2,\varepsilon_1,\varepsilon_2)\\
f(\sigma_1|\mathbf{D},\mu_1,\mu_2,\sigma_2^2,\varepsilon_1,\varepsilon_2) &\propto f(\mathbf{D}|\mu_1,\mu_2,\sigma_2^2,\varepsilon_1,\varepsilon_2)f(\mu_1,\mu_2,\sigma_2^2,\varepsilon_1,\varepsilon_2)\\
f(\sigma_2|\mathbf{D},\mu_1,\mu_2,\sigma_1^2,\varepsilon_1,\varepsilon_2) &\propto f(\mathbf{D}|\mu_1,\mu_2,\sigma_1^2,\varepsilon_1,\varepsilon_2)f(\mu_1,\mu_2,\sigma_1^2,\varepsilon_1,\varepsilon_2)\\
f(\varepsilon_1|\mathbf{D},\mu_1,\mu_2,\sigma_1^2,\sigma_2^2,\varepsilon_2) &\propto f(\mathbf{D}|\mu_1,\mu_2,\sigma_1^2,\sigma_2^2,\varepsilon_2)f(\mu_1,\mu_2,\sigma_1^2,\sigma_2^2,\varepsilon_2)\\
f(\varepsilon_2|\mathbf{D},\mu_1,\mu_2,\sigma_1^2,\sigma_2^2,\varepsilon_1) &\propto f(\mathbf{D}|\mu_1,\mu_2,\sigma_1^2,\sigma_2^2,\varepsilon_1)f(\mu_1,\mu_2,\sigma_1^2,\sigma_2^2,\varepsilon_1)
\end{aligned}
$$

MCMC includes methods and algorithms that allow sampling from a distribution $f(x)$ even though only a proportional distribution $g(x) \propto f(x)$ is known. A set of priors is needed to start the iterations,

$$\{\mu_1^0, \mu_2^0, \sigma_1^{2^0}, \sigma_2^{2^0}, \varepsilon_1^0, \varepsilon_2^0\}$$

For each parameter the conditional distribution is sampled using the initial values chosen as priors, and in each iteration the parameter values are changed. After a number of iterations the algorithm will start to converge in distribution. The distribution found in convergence of the parameters is the final distribution estimate. The initial period before convergence is called the burn-in period, which varies in size depending on the problem.

2.5 Nonlinear modelling using Direct Look Up

To extend the the Direct Look-Up method for mixture distributions it is important to establish the original method of DLU.

The Direct Look-Up method (DLU) is a method developed by Isaeva et al. [2011a,b] for estimating nonlinear mathematical functions. It is developed for use on curves in the range $[0, 0]$ to $[1, 1]$, having only one inflection point.

The following is a descriptive summary of the DLU method. The general idea behind this method is to construct a library of curves, computed for a dense grid of the possible parameter values for the library function. Library curves are found for each combination of the parameter values. The curves are then stored for easy retrieval when performing look-up of new data. When estimating new data, the data is looked up in the curve library, and the curve with the closest fit is chosen. This is a noniterative process, and there is no risk of the method not converging to a final estimate. Generating the curve library takes time to perform, but this

is not necessary for every estimation problem, the look-up uses a saved library. This allows testing of multiple nonlinear mathematical functions without any previous knowledge about what model might be behind the observed data.

In Isaeva et al. [2011a,b] 38 nonlinear mathematical functions were chosen to be included in the library. For every model $F_m(x; \mathbf{p}_{j,m})$, N_m curves were generated as

$$z_{j,m}(x) = F_m(x; \mathbf{p}_{j,m})$$

where x is 100 time points from 0.001 to 1, $\mathbf{p}_{m,j}$ is the parameter set for simulation j and model m.

The transformation:

$$y_{j,m} = \frac{z_{j,m} - off_{j,m}}{sl_{j,m}} \tag{2.12}$$

was performed on all curves to achieve the limits

$$y_{j,m}(0.001) = 0 \quad \text{and} \quad y_{j,m}(1) = 1$$

The parameters $off_{j,m}$ and $sl_{j,m}$ are the offset and slope parameters respectively to achieve the limits.

In cases where the x-axis is not scaled between 0 and 1 this also needs to be transformed by:

$$\mathbf{x} = x_{min} + (x_{max} - x_{min}) \frac{\mathbf{x}_{input} - x_{min_0}}{x_{max_0} - x_{min_0}} \tag{2.13}$$

The values x_{min} and x_{max} are the minimum and maximum values of the desired x-axis. The values x_{min_0} and x_{max_0} are the minimum and maximum values of the input observations x-axis, $\mathbf{x}_{input}$.

Each curve is recorded with all parameter variations, while still following the set limits, and stored in a database of function values. For functions with less than 4 parameters, the function line was found over a fine grid of evenly spaced parameter values. Functions with more than 3 parameters were run after a Multi-level Binary Replacement (MBR, see section 2.6) design, because of the combinatorial expansion that happens for many factors with multiple parameters.

A Principal Component Analysis (PCA, see 2.8) is then performed on this database of function values to find the following compression

$$y_m = \bar{y}_m + T_m V_m' + E_m$$

where T_m and V_m are a set of scores and loadings of the data, and E_m are the residual values not explained in the set of scores and loadings. Instead

of storing the complete curve library of Y, it is reduced into a vector containing the arithmetic means, a $n \times k$ matrix in the scores and a $k \times k$ matrix for the loadings and a vector of errors. Here k is the number of principal components chosen to be included from the PCA. The variation in the curve library can be restored to a certain degree by these k components contained in the scores T_m and loadings V'_m. The amount of data points to be stored are thus drastically reduced compared to the entire curve set.

To help assess the new parameter estimations residuals are found for each curve $y_{j,m}$ in the model by

$$e_{j,m} = (y_{j,m} - \bar{y}_m) - t_{j,m}V'_m \quad \text{for model m, curve j.}$$

where $t_{j,m}$ is the score vector of curve j, model m.

A mean sums of squares of errors is then found from the residuals as

$$s^2_{j,m} = \frac{e_{j,m} \cdot e'_{j,m}}{K}$$

where K is the number of observations per curve, in this case 100.

For each curve in the library the errors were recorded and different percentile limits were found. The percentiles define a range for expected errors.

To perform look-up estimation of N new observations of the general form

$$z_i^{obs}(x) = a_i + b_i F_m(x; \mathbf{p}_i) + e_i(x) \tag{2.14}$$

where a_i and b_i are the offset and slope parameters describing the difference between the observed curves and the standardized function curves. Here a_i is the minimum value of $z_j^{obs}(x)$ and b_i the difference between the maximum value and the minimum value.

The processed observation curves are found as

$$y_i^{obs} = \frac{z_i^{obs} - off_i^{obs}}{sl_i^{obs}}$$

when the offset off_i^{obs} and slope sl_i^{obs} are chosen to achieve $y_i(0.001) = 0$ and $y_i(1) = 1$.

The new curves are projected onto the library as

$$t_{i,m} = (y_i^{obs} - \bar{y}_m)V_m$$

with the residuals and errors found as

$$e_{i,m} = (y_i^{obs} - \bar{y}_m) - t_{i,m}V_m' \quad \text{and} \quad s_{e_{i,m}}^2 = \frac{e_{i,m} \cdot e_{i,m}'}{K}$$

The projection of new curves means estimation without iterations. There is no iterative algorithm deciding the output parameters, no dependency on the solution reaching convergence.

In Isaeva et al. [2011a,b], only the curves where $s_{e_{i,m}}^2$ less than the 99% confidence interval limit for the $s_{e_{j,m}}^2$ of the library curves $y_{j,m}$ was considered for parameter estimation. The new curves are compared to every curve in the library, and the residuals and errors are computed. Comparing the errors to the percentile for error in the curve library, the library curves within the limits for natural error in the library were chosen.

The distance is computed in the score space for the selected curves using ordinary Euclidean distance

$$s_{t_{i,m}} = \sqrt{(t_{i,m} - t_{j,m})(t_{i,m} - t_{j,m})'} \quad \text{for} \quad i \neq j$$

After sorting according to error and distance, a set of possible parameter estimates is found. Only the removal of the preprocessing is then necessary, to find a_i and b_i from 2.14.

$$\hat{a}_i = off_i^{obs} - off_{j,m}\frac{sl_i^{obs}}{sl_{j,m}} \quad \text{and} \quad \hat{b}_i = \frac{sl_i^{obs}}{sl_{j,m}} \tag{2.15}$$

The estimated equation and parametrization is then given as

$$\hat{z}_i^{obs}(x) = \hat{a}_i + \hat{b}_i F_m(x; \hat{\mathbf{p}}_i) \tag{2.16}$$

Instead of now deciding what model is the indisputably best solution, the e.g. 10 best solutions are reported back. This allows the user to decide whether or not the best or one of the best seems a plausible solution. The user might have previous knowledge regarding what type of function it might be, and may choose another than the apparently best fit.

Reporting the 10 best solutions is both a strength and an issue with this method, as not giving a concluding answer might not be of so much use to an inexperienced user. Other issues that arise with this method are that there are not that many problems that can be described by just one function, they are often a sum of several functions or distributions and will need more complicated models.

Another problem is that the more curves there is in the library, the longer the curve fitting takes. Choosing the nearest neighbour solution as is

done when finding the least error, works very well if the space is densely sampled, but it also increases the number of curves drastically. Another technique may be used to find the best parameter set after the best function is found.

Statisticians may miss the ordinary possibilities for significance testing and error estimation which are not readily available with the DLU. To get an evaluation of the estimated solution another method will have to be implemented. For instance; find the boot-strap or cross-validation estimate of the traditional statistical fittings. (Hastie et al. [2009, chapter 7] contains more info on these methods)

This is based on the development stages of this method, and is all found in Isaeva et al. [2011a,b].

2.6 Multi-level Binary Replacement Design

During the implementation of Isaeva et al. [2011a,b] the need for an experimental design method for creating Design of Experiments (DoE) for problems with many levels in each factor became apparent. For mathematical models with more than three parameters the number of unique combinations of factor levels became too large for it to be possible to run every combination. Traditional DoE works on factors with few factor levels, to create a set of runs that combines all possible factor levels for different experiment settings. As the mixture models solutions space is spanned by 5 factors, or model parameters, each with multiple levels, Multi-level Binary Replacement (MBR) design is a good choice for reducing the number of unique combinations to include in the curve library for the mixture models. MBR design allows creating a fractional factorial design, drastically reducing the number of runs for design factors of multiple levels.

Classical Design of Experiments considers a situation where the measured outcome of an experiment is supposed to be a function of the expected value of the factors being examined. To study the effects of changes in factor levels, experiments are performed. The experiments can show what effect different factors have on the response. The challenge in all DoE problems is choosing which values of the factors to be run in an experiment.

The basic goal of DoE is to set up a design for performing experiments. DoE defines the levels of each factor, which levels to be run and the order of the runs. The design combines the factor levels, allowing the study of main effects and interaction effect of the factors. It is however apparent that this will lead to an extraordinary large amount of combinations when

the number of factors get large or each factor has multiple levels. Most experiments have a cost, and in the case where there are many factors and levels, though possible, it is often desirable to reduce the number of runs. In many situations the higher order interaction terms are of negligible effect compared to the main effects, and they can be assumed to have no effect. Main effects can then be confounded with higher order interactions to reduce the number of runs. When terms are confounded with each other it means that there is no possible means of identifying what part of the effect comes from one or the other, but when assuming that the higher order interactions are 0, this is safe.

Reducing the number of runs by confounding factors is achieved through a fractional factorial design. A fraction of the original run are used, creating a smaller design. The standard methods for doing this is developed for two-level factorial designs.

A fractional factorial design (e.g. see Montgomery [2009, chapter 8.4]) reduces the number of runs necessary by confounding higher order interactions with main effects. In a general 2^{k-p} fractional design there are 2^{k-p} runs. For instance, if $k = 4$, with factors A, B, C and D, but the experiment is so expensive that only eight experiments can be performed, a fractional design is created. The fractional design let us avoid doubling a 2^3 full design for A, B and C to include D. To achieve eight runs, p must be one. In a 2^{4-1} fractional design there must be one generator. A generator is the defining factor, the combination of the other factors that combined will confound with the factor D. One possibility for this instance is to take the generator for $D = ABC$. The number of generators is equal to p. In DoE terminology this gives that

$$\begin{aligned} D &= ABC \\ ID &= ABCDD \\ I &= ABCD \end{aligned}$$

since the contrasts are orthogonal and $DD = I$. The main effect of D is confounded with the third order interaction ABC. That means that there is no possible way to discern what effects are due to the interaction effect and what is because of D.

The design schema then becomes

	A	**B**	**C**	**D = ABC**
(1)	−	−	−	−
a	+	−	−	+
b	−	+	−	+
ab	+	+	−	−
c	−	−	+	+
ac	+	−	+	−
bc	−	+	+	−
abc	+	+	+	+

As can be understood from the combinatorics of increasing the number of factors, the number of runs can get huge.The method also only supports orthogonal contrasts, where each variable has only 2 levels. In DLU, however, there are usually many levels per variable. This is addressed in Martens et al. [2010] to solve these issues.

The idea as presented in Martens et al. [2010] is to recode the design factor x_k using bits, to give every factor level of x_k a binary code. The values of the design factors need not be equally spaced or have the same values for all variables, and to simplify they are all recoded into a decimal indexing variable d_k. The binary number system consists of only two numbers, 0 and 1. Every decimal place represents a factor of two, as opposed to a factor of ten in the ten number system commonly accepted. The binary code representing this value can then be used as separate binary factors for design of experiments. The MBR method assumes that each factor d_k have $L(k) = 2^{M(k)}$ number of levels, when $L(k)$ is the length of factor d_k.

For each design factor's index variable the levels are recoded into $M(k)$ factor bits $[f_{k,1}, \ldots, f_{k,M(k)}]$. As an example, if $L(k) = 4$ then $M(k) = 2$ and the factor levels are $\{0, 1, 2, 3\}$. The factor variables can be recoded into binary bit variables. As $M(k)$ increases, the number of bits in the bit variable increases. The one factor with four levels would be recoded into two bit factors, f_{k_1} and f_{k_2}, which make up the bit factor matrix F_k.

$$d_k \rightarrow f_{k_1} f_{k_2} = F_k$$

$$\begin{bmatrix} 0 \\ 1 \\ 2 \\ 3 \end{bmatrix} \rightarrow \begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix}$$

This gives the basis for binary replacement. Each factor design variable has been recoded into a binary value, and the binary values are split into separate factors. This transformation allows large fractional factorial designs to be created from factors with multiple levels. A weakness is that the length of a design factor must be a factor of 2, to fulfill the definition of length $L(k) = 2^{M(k)}$ of a factor.

To use the bit factors for DoE, they can be shifted to the traditional $1/-1$ values normally used to create designs. A design factor k then becomes

$$d_k = \begin{bmatrix} 0 \\ 1 \\ 2 \\ 3 \end{bmatrix} \Leftrightarrow F_k = \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix} \Leftrightarrow G_k = \begin{bmatrix} -1 & -1 \\ -1 & 1 \\ 1 & -1 \\ 1 & 1 \end{bmatrix} \tag{2.17}$$

where F_k is the factor matrix representing factor k, and G_k is shifted for use in the experimental runs.
Turning the run scheme back from bit factors to the original factors is simply done by combining the right bit factors again. Since every set of bit factors represent a index value in the original factor this is straight forward.

2.6.1 Confounding issues

Confounding is a technique used in DoE to arrange experiments in a way to reduce the number of runs. It causes some of the experiments information to be indistinguishable from other parts of the experiment, see e.g. Montgomery [2009, p. 275]. In all fractional designs there are confounding factors, that is the trade off for reducing the number of runs. The confounding factors can be worked out from the defining relations and the generators, finding both the number of factors and which factors that are included.

Confounding issues are difficult with the MBR design method. Normally, one will choose generators to optimize alias structures and other criteria for design optimality to avoid large unintentional patterns in the confounding factors. With the replaced factors, however, the bit factors are not very representative of the spatial spanning of the design space on their own, and it is difficult to predict the effect of the confounding patterns. After the binary replacement, the bit variables are confounded with other bit factors, where all represent magnitudes of a design factor.

Strategies to avoid too much confounding in one design factor compared to the other factors could include visual inspection after transforming the bit factors back to the design space, careful choosing of the generators or selecting optimal designs by optimizing criteria for DoE problems. Different

theories for confounding patterns are proposed, choosing confounding patterns from bit factors from the same design factor, not including any bits from the same design factor or letting the corresponding design factors confound with each other or not.

Looking at the example from earlier, with $k = 4$ factors, and using $D = ABC$ as a generator for a 2^{4-1} fractional factorial design, then the aliasing structure giving the confounding factors is:

$$AD = BC$$
$$DC = AB$$
$$AC = DB$$
$$A = BCD$$
$$B = ADC$$
$$C = ADB$$

So if A and B are bit factors for one four level factor and C and D for another four level factor, we observe that, for instance, the bit factor interaction of AB is aliased with the bit factor DC. It is not clear from the alias structure at the bit level how the confounding patterns will turn out at the level of the original factors.

Every main effect confound with the third order interaction including the other factors and all combinations of two order interactions are confounding.

2.6.2 Generators

A 2^{k-p} fractional factorial design requires p generators. Generators are the combinations of other factors to make up the confounding factors. The example stated above has three individual design factors, while the fourth factor is defined as a combination of the first three. The generator is specified as $D = ABC$, which allows the experiment to be run using a 2^{4-1} design, reducing the number of individual runs to $2^3 = 8$ from $2^4 = 16$. For a bigger set of factors where the fraction is larger than one, i.e. 2^{5-2}, two generators have to be defined in order to run the experiment. There would be three independent factors and two that are combinations of the first three.

Generator sets are known in literature and software for reasonably large problems. The magnitude of the problems that have MBR applied can be much larger than this, leaving the known generator sets useless.

Generators have to be specified for the design. In Martens et al. [2010] a set of alternating generators and confounding patterns have been generated,

and a design has been chosen after visual inspection of the confounding patterns, to confirm that the chosen design spans the spatial room of the design factors.

Traditional knowledge about resolutions and aliasing structures does not give the same meaning when the bits are confounding instead of the spatial design factors.

2.6.3 Optimizing Criteria

Rather than using visual inspection of the designs spanning capabilities, optimizing techniques for choosing optimal designs exist. There are different ways of optimizing the DoE. In fractional design optimizing is done by choosing the generators and confounding structures that will optimize one or more chosen criteria. Good designs and generator sets are recorded in books and articles, and many of the statistical softwares have these built in and ready for use. However, for the MBR design there are no previously known optimized structure, because of the size of the problem and the transformation to bit factors.

There are many interesting possibilities for optimizing the design generators. Some of these have been explored in a master thesis at NTNU by Thalberg [2011].

To define the optimality criteria for design generators we need the design matrix G_k from the MBR design process and its information matrix, $M = G_k' G_k$.

Finding the A, D and E optimality is done by

- A-optimality: minimize the trace of M^{-1}
- D-optimality: maximize the determinant of M
- E-optimality: minimize the largest eigenvalue of M
- V-optimality: minimizes the average prediction variance
- G-optimality: minimizes the maximum diagonal element of the design matrix's hat matrix, thus minimizing the prediction variance
- Condition number: minimize the condition number of the design matrix

The three first criteria check the design matrix. These methods may not work well with the binary replacement, as transforming the design back

into the design factor may change the optimality. A criterion that optimizes for the prediction variance or the final design matrix after replacing the binary values into the design space will allow for optimization in the design space, rather than the bit space. V-optimality optimize the design space, not the bit space.

In Tøndel et al. [2010] the MBR method is optimised using one criterion to decide between different types of design and another criterion for choosing a final design within that type. Optimisation was performed by randomly selecting generators and design, transforming back to the design space and calculating an optimisation criterion on the design factors. Optimality criteria that were used to optimize MBR was V-optimality and the condition number.

In Thalberg [2011] the MBR method is optimised using residual mean square error and MAX criteria, which choose designs based on the RMSE and the maximum error. Comparison studies there show that the different confounding patterns score best in different optimizing criteria and for different model functions, and that confounding patterns and criteria should be chosen depending on what type of function the model is estimating.

2.7 Estimation of prediction error

When fitting statistical models an indication of how well the model can predict new outcomes is wanted. Risks concerning uncertain models include parameter estimates that vary greatly or poor generalization to other data not used to fit the model. The prediction error and how to estimate it is therefore important knowledge relevant for choosing models and parametrizations.

As is shown in Hastie et al. [2009, 7.2] the generalization error, or test error is expressed as

$$Err_\tau = E[L(Y, f(x))|\tau] \tag{2.18}$$

where τ is an independent test set and $L(Y, f(x))$ is the loss function used to find the error. The function $f(x)$ is the predictor for Y. The loss function is dependent on the model to find errors for, but often on the forms of

$$L(Y, f(x)) = (Y - f(x))^2 \quad \text{or} \quad L(Y, f(x)) = |(Y - f(x))|$$

Loss functions are measures of the distance between observed values and estimated values. Other loss function include loss on log likelihood, such as

$$L(Y, \hat{p}(x)) = -2\log\hat{p}_Y(x)$$

which is used for modelling probabilities. Then $\hat{p}(x)$ is the estimated probability function for Y, and $\hat{p}_Y(x)$ the estimated probability of Y given data x.

The absolute value or the squared of the value are used instead of the distance or the residual itself, to avoid all the distances averaging to 0. Err_τ is the prediction error over an independent test sample, and this is the prediction error of interest for evaluating the prediction capabilities of a model.

The expected test error is the expected value of the equation 2.18, and it averages over all that is random, including the test sample.

The training error is the average loss over the training sample.

$$\overline{err} = \frac{1}{N}\sum_{i=1}^{N} L(y_i, \hat{f}(x_i)). \tag{2.19}$$

This is easier to find with statistical computation methods than the test error, however it is not a good estimator for the test error. This estimate of the error will decrease steadily as the model complexity increases, fitting the model closer and closer to the data. If the model is fit too closely to the data, the generalized test error will get very large, as it will fit very poorly for any data other than the training data.

In situations where there are much data the solution that will both select the best model and have the least general prediction error is to split the data into a Training set, a Test set and a Validation set. The training set is used to fit the model and find the training error. The validation set is used to estimate prediction error for model selection, and the test set is used only for generalization assessment of the final, chosen model. The data in the test set should ideally not be used more than on the final chosen model. This ensures that data is not too closely fit to the data for poor generalization.

2.7.1 RMSEP

The Root Mean Square Error of Prediction is estimated as the square root of the mean square error for prediction. It is found by using the normal euclidean distance between the predicted value and the observed value as the loss function.

For a model $Y = f(X) + \varepsilon$ the expected error of prediction at a point $X = x_i$ can be found using the squared loss function as

$$Err(x_i) = E((Y - \hat{f}(x_i))^2 | X = x_i) \tag{2.20}$$

and it is shown in Hastie et al. [2009, chapter 7.3] that this can be split into three parts,

$$Err(x_i) = \sigma^2 + Bias^2(\hat{f}(x_i)) + Var(\hat{f}(x_i)) \tag{2.21}$$

The first part is an irreducible error, the part that cannot be reduced. This is the variation of Y around $f(x)$. The bias term represents the errors that are done in simplifying into the current model situation. This is generally decreased with model complexity or using unbiased methods. The variance part is the prediction variance at the point in question. In total this becomes the prediction error at x_i.

Using the log likelihood function as the loss function leads to measures such as the Akaike Information Criterion (AIC) and the Bayesian Information Criterion (BIC) which provide an estimate of the test error. See Hastie et al. [2009, chapter 7.5,7.7] for details. Methods such as cross-validation, see Hastie et al. [2009, chapter 7.10], and boot-strap sampling, see Hastie et al. [2009, chapter 7.11], provide estimates of the prediction error Err directly.

2.7.2 Log Likelihood comparison of method performance

Using the log likelihood function as a loss function for the in sample error works for a wide range of model estimation methods. Model selection criteria for the log likelihood loss function, such as AIC and BIC, penalize complex models, to avoid overfitting the data to a overly complex model. For comparing models of equal sizes this is not a problem, as the the model have the same number of parameters, and the relative sizes between the models is what is important.

As presented in Dias and Wedel [2004] the log likelihood of a model estimate can be used to compare different methods for model estimation. The maximized log likelihood or loglik is a measure of how well the model fits with the observed data. The likelihood function is often a complicated expression of the probability of every single data point given the function parameters, assuming that the function values are independent of each other.

$$l(\mathbf{x}) = f(x_1)f(x_2)\dots f(x_m) = \prod_{i=1}^{m} f(x_i) \tag{2.22}$$

for all $i = 1, 2, \ldots, m$. The reason for using the log of the likelihood value is to simplify this, as the log of a product is a sum of the log values of the product terms. The log likelihood for a likelihood function such as in 2.22, there are

$$loglik = log(l(\mathbf{x})) = log(f(x_1)f(x_2)\ldots f(x_m)) = \sum_{i=1}^{m} log(f(x_i)) \qquad (2.23)$$

where $f(x_i)$ is the density for x_i.

There are concerns to address when using the log likelihoods for model comparisons. In other modelling situations there is a need for a model selection criteria. These take into account the model size, the number of parameters in the model, to be able to compare models of different sizes. Comparing results from EM, MCMC and DLU however, no such criteria is needed as they are all estimating the same model. The log likelihood values can be compared directly.

A higher value for the log likelihood is equivalent to a better fit. Finding the log likelihood value is finding the probabilities of the observed data given the parameters that have been estimated. For some methods a log likelihood result can be higher for an estimated model than for the true underlying parameter values, depending on the sampling density and sample size of the input data, and on how prone the estimation method is to overfit the model to the specific data.

2.8 PCA

Principal Component Analysis (PCA) is a method used for data exploration. It explains the structure of the variance and covariance matrices of a set of variables by a set of linear combinations of these variables. To explain all the variability of p variables, p linear combinations are needed, but in many multivariate situations most variation can be explained by a reduced amount of linear combinations. These linear combinations are called principal components. If almost all variation can be explained by k principal components, they can replace the original variables, reducing the data from n observations by p variables to n observations by k principal components. See Johnson and Wichern [2007, chapter 8] for details.

Analysing the principal components will show what variable has the most influence on the variability of the data. Each component will identify a direction of the variation in the data.

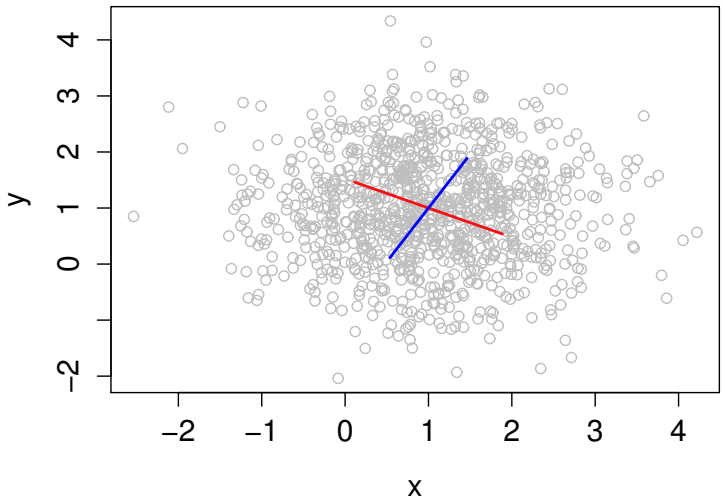


Figure 2.4: An example of multivariate data, little correlation and no obvious strong direction

To illustrate the meaning of direction of variation, see the figures 2.4 and 2.5. One situation, there is clearly one major direction, which will be the first principal component (PC). The second one will be of much less importance. Another situation there might not be any clear direction to the variance at all, and the variation can not be explained by less than all the data, it cannot be reduced by PCA.
Each PC will be orthogonal onto the previous one. In short, the first PC is the direction of largest variation in the data. The second PC is the direction of most variation, orthogonal to the first component. The third component is orthogonal to both the first and the second component.

There are several strategies finding the principal components. Using the covariance matrix Σ for data X, the principal components are found using the eigenvectors and eigenvalues of the covariance matrix. Let $[(e_1, \lambda_1), (e_2, \lambda_2) \dots (e_p, \lambda_p)]$ be the eigenvalue eigenvector pairs of Σ. Then the i'th principal component is found as $e_i' x$.

Singular value decomposition (SVD) is an alternative for PCA computed from the eigenvalues. SVD states that for any $n \times p$ matrix A with rank r, there exists a decomposition such that

$$X = USV' \tag{2.24}$$

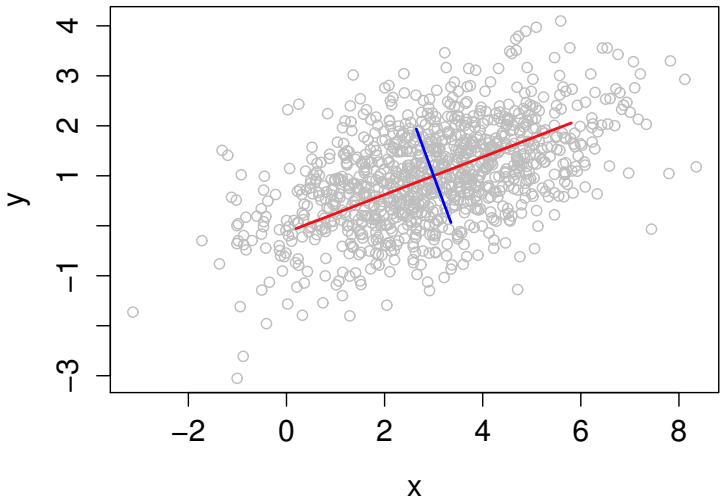


Figure 2.5: An example of multivariate data, high correlation and strong direction

where U is a $n \times n$ matrix containing the left singular vectors of X and V is a $p \times p$ matrix containing the right singular vectors of X. S is a $n \times p$ matrix containing the r singular values of X. (Theorem 10, Lay [2006, p.491])

If the principal components are computed from the covariance matrix of the centred X, the matrix $X'X$ is proportional to the covariance matrix of X, see Wall et al. [2003]. The singular values squared is proportional to the variance of the principal components.

To find the principal components from the covariance matrix, the covariance matrix is diagonalized. The resulting matrices contains the loadings, scores and the variances of the components. The right singular vectors of V are contain the loadings. A principal component is a random variable, where the loadings define the linear combination of the original variables. The squared singular values are proportional to the variance of the principal components. The left singular vectors, U, multiplied by the diagonal matrix S is called a scores matrix. The scores are the data multiplied by the loadings of the principal components, giving the transformed value in the principal component space.

$$X = USV'$$

$$XV = USV'V = US$$

Evaluating new data in relation to the principal components is done by transforming the new data into scores by multiplying with V.

The amount of variation explained by each principal component can be found using the squared singular values. The σ_i^2 are the variances of each principal component and the proportion of variance explained by component number i can be expressed as

$$\frac{\sigma_i^2}{\sigma_1^2 + \sigma_2^2 + \cdots + \sigma_p^2} \quad \text{for all} \quad i = 1, 2, \ldots, p \tag{2.25}$$

For more detail on PCA, see Johnson and Wichern [2007][p:430-441], on SVD see Lay [2006][p:487-496] and on the connection between the two see Wall et al. [2003].

In Wall et al. [2003] there is an interesting mathematical connection between the SVD and the Fourier expansion. The singular value decomposition used to find scores and loading for the curve library can be expressed as

$$Z = USV' = \sum_k \mathbf{u}_k s_k \mathbf{v}_k^t$$

Looking at the Fourier expansion of a matrix:

$$z_{ij} = \sum_k c_{ik} e^{\mathbf{i}2\pi jk/m} \tag{2.26}$$

By normalizing $e^{\mathbf{i}2\pi jk/m}$ and calling it $\mathbf{v}'_k$ the following is achieved

$$z_{ij} = \sum_k b_{ik} v'_{jk} = \sum_k u'_{ik} s'_k v'_{jk} \tag{2.27}$$

This shows the similarities between the SVD and the Fourier transform, even if the components are not generally the same.

Principal component analysis may also be used as a tool for data reduction. If p is large and there is reason to believe that all the variables are not needed to express the changes in data, the principal components can be used to reduce the number of variables. If for instance the five first components are enough to capture 99% of the variation in the data, the five first scores and loadings vectors are enough to reconstruct the original data to 99% of the original variation. Statistical models can now be used on these five new variables instead of the original p variables. This means that

all calculation is moved into the score space, but is transformed back to the original dimensions after using the model.

Reducing the variables is desirable to reduce the amount of data that needs to be stored and handled, creating faster programs.

Finding the number of principal components to retain for analysis is a matter of discussion, and there is no definite answer to this. It depends on the need of precision in the final results and how much of the variability in the original data is needed for the results to be usable. The amount of variation explained by each principal component might drop steadily per component or stabilize after a few components. If the amount stabilizes, very many extra components is needed to explain more of the variance, and it can be discussed whether or not to include more variables. For instance, if 98% of the variability is explained by the 5 first principal components, but 10 more is needed to achieve 99%, it might be up for discussion whether or not to include these 10 extra components or make due with only 98% explained. If the right balance between a high level of precision and fewer variables to explain the variability a compression of data can be achieved, saving both memory and computing time for large situations with much computation. The goal is to take k sufficiently large and conveniently small.

2.9 Software

For the implementation of code and simulation of all experiments, the free software R is used. This has the advantage of being free, easily extended through user community packages and is always revised, updated and supported. It works on a wide range of computer platforms without needing conversion from one operating system to another. For more information regarding development and use of R, see R Development Core Team [2011].

This thesis is written using LaTeX, a free software for accessing the powerful typesetting tools in TeX. TeXis the typesetting system released in 1978 by Donald Knuth. These are widely used in academic writing.

Chapter 3

Method - DLU for mixture distributions

Estimating the parameters of the mixture distributions is an area where DLU can be applied. It avoids the convergence issues, start value dependency and the iterative processes connected with the traditional estimation methods for mixtures. The DLU idea is to pre-compute the mixture distribution values over a dense grid of parameter values, then compress the solutions using PCA to store only the necessary information for later use. Parameter estimation for new mixtures can then be performed very quickly, without the ordinary issues of parameter estimation.

Parametric estimation of mixture distributions represent a difficult estimation problem, with an endless number of possibilities. To start the implementation a few choices had to be made regarding the size of the problem and the parameters to be estimated. There is an infinite number of options when it comes to what combinations could be created and how many elements a mixture should consist of. The mixture, the parameters and the design of the library needs to be defined. Also, there are two options for what functions to use in DLU, either the density function $f(x)$ or the distribution function $F(x)$.

3.1 Mixture of normal distributions

Firstly, the scope of the problem must be set. The mixture of two normal distributions is chosen for the DLU. Looking at a mixture model as defined in section 2.3,

$$f(x) = \varepsilon f_1(x) + (1 - \varepsilon) f_2(x)$$

there are two components to this.

This is defined on a set of values, chosen as

$$x \in [0.001; 1] \tag{3.1}$$

Distributions defined on other intervals can be transformed into this range and back again after estimation, the limited range is only for estimation purposes.

It was chosen to implement a mixture of Normal Distributions, as in equation 2.1. The normal distribution has two parameters, the expectation μ and the variance σ^2. The variable X is distributed as

$$X \sim N(\mu, \sigma^2)$$

A mixture of two normal distribution, each with its two parameters, will lead to five parameters in total for the mixture, including the weighting or mixture parameter, ε.

These are $\mu_1, \mu_2, \sigma_1^2, \sigma_2^2$ and ε and the model function will be

$$f(x) = \varepsilon N_1(\mu_1, \sigma_1^2) + (1 - \varepsilon) N_2(\mu_2, \sigma_2^2) \tag{3.2}$$

To look at an example of a variable that follows a mixture of normals, look at the heights of individuals in a population. If height is recorded independently of gender, the height would follow a mixture distribution, with one component representing each gender. The mixing parameter would describe the number of males compared to females.

DLU works by first calculating the mixture distribution for every combination of parameters over a dense grid of parameter values, covering the entire parameter space. This requires knowing which combinations of parameters for the two-component mixture model are to be included. A curve library containing these mixtures is the goal. The library must be large enough to give a satisfactory dense coverage of the parameter values. The density of the data points needs to be sufficiently high to allow for accurate estimation, since a weakness of this method is that the only parameter values that will be predicted are ones already found in the curve library.

To illustrate the density of the data points and the effects in combining parameters two example parameters μ_1 and μ_2 are compared. As the DLU method will be limited to the parameter values that are already in the database, the parameter density affects the result of the parameter estimation in DLU. The higher the density of the parameters, the more

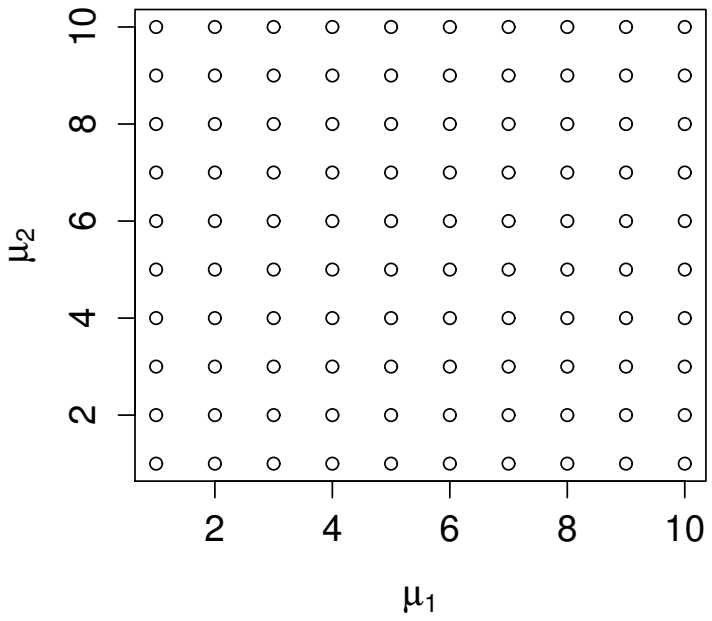


Figure 3.1: Illustration of possible combinations of example parameters μ_1 and μ_2 at a low number of levels for each parameter.

combinations are in the database as well, requiring more runs, or more confounding factors using a MBR design.

The example figure presented in 3.1 show that for the spatial distance between each point of the example parameters μ_1 and μ_2 are much larger than for the the parameter values showed in figure 3.2. From the low to the high density spatial coverage, there are twice as many points of each of μ_1 and μ_2. The number of combinations are four times as many as for the low parameter value density, and four times as many points in total. This can be extended to cover virtually every discernible value combination in the parameter space, but with a cost of runtime, storage and computational power.

The density of the parameters and number of curves to include in the library was also an issue in Isaeva et al. [2011b]. There it was chosen that for functions with more than three parameters, multi-level binary replacement design was performed to create a fractional design of

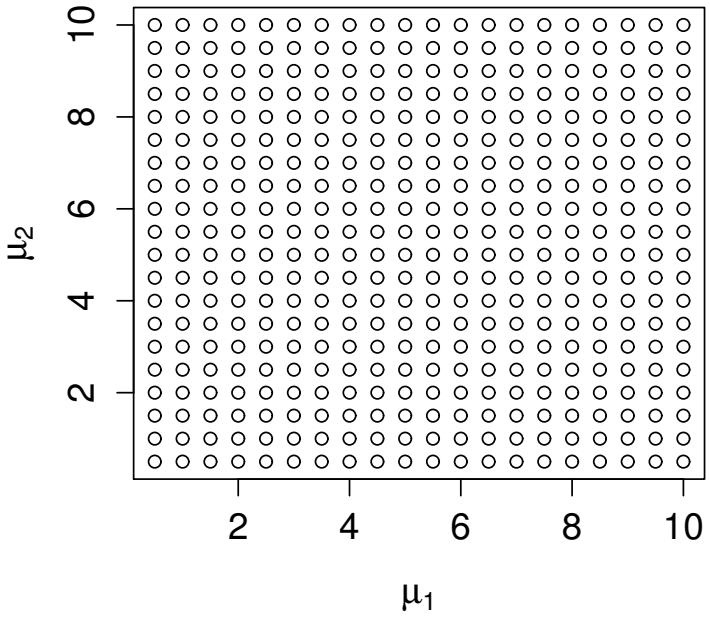


Figure 3.2: Illustration of possible combinations of example parameters μ_1 and μ_2 at a high number of levels for each parameter.

experiments and reduce the number of runs. This allows a dense sampling of the parameter values for yet retaining high accuracy in the parameter estimation.

3.2 Parameters and MBR

The levels of all parameters in the database need to be defined. Using an x range as defined in equation 3.1 sets the range of all distributions. All distributions to include in the curve library are defined in this range, and any new observations need to be transformed to this range to be able to use the DLU method for estimation. Sampled data or observational data are transformed using a density function estimate to find the density estimate $\hat{f}(x)$ on a given number of points, here for the 100 data points of x. The density function estimate is from the R package stats, see R Development Core Team [2011]. It is a generic function that estimates the kernel density of the data.

There are in total five parameters of the mixture of two normal distributions. The first parameter, the mixing parameter ε describes the weighting of the two distributions. This also makes up a weighted sum. The weighted sum limits the function to follow the rules for distribution functions, that the total area under the curve of the density function $f(x)$ will always be equal to one, $\int_0^1 f(x)dx = 1$, and the limits$f(x < 0.001) = 0$ and $f(x > 1) = 0$, while for the distribution function $F(x)$ the limits $F(0.001) \sim 0$ and $F(1) = 1$ are kept at all times. This is to make the density and distribution function valid probability functions on the interval of x.

When combining two normal distributions, one half of the functions will be mirror images of each other. Hence, the mixing parameter ε is limited to $0 < \varepsilon < 0.5$ in the library. To have a set of defining rules to keep all lines and functions to the same area of values is essential for the method to be useful. A set of rules defining the parameter values to keep the function as a distribution function on $x \in [0.001; 1]$ were defined as follows.

The expectations are in the interval $0.01 < \mu_i < 1$ for all μ_i. Minimum value of 0.01 is to ensure that parameter combinations where one or more of the expectations are almost zero do not happen.

To achieve $F(0.001) \sim 0$ and $F(1) = 1$ and $\int_0^1 f(x)dx = 1$ some approximation is done, but limits for cut off must be defined. The expectation plus 3 standard deviations has to be less than 1 at the same time as the expectation minus 3 standard deviations has to be larger than zero. This ensures that there is a very small area that falls outside of the limits given by x.

The relationship between μ_i and σ_i needed to achieve this is describes as

$$\mu_i - 3\sigma_i > 0 \text{ and } \mu_i + 3\sigma_i < 1$$

which translates into the following requirements for the σs:

$$\sigma_i < \frac{\mu_i}{3} \quad \text{and} \quad \sigma_i < \frac{1 - \mu_i}{3}$$

These requirements limit the parameter space somewhat.

In the curve library the σ_i is chosen to be larger than 0.001 and still fulfill the requirement.

The implementation of MBR (see 2.6) design is chosen to reduce the number of curves that needs to be generated. This allows the creation of a fractional factorial design, see e.g. [Montgomery, 2009, tchapter 8], to reduce the number of runs.

Creating a curve library and a DoE for a continuous parameter situation requires some compromise. Each parameter needs to be approximated using discrete values.For each parameter the range of values were distributed into a number of equally spaced values, giving a discrete approximation to the continuous parameter values. Even with the discrete approximation, to ensure the prediction accuracy the samples need to be dense to achieve good results. The number of different parameter values can get large, and so the combined number of the combinations of all chosen parameter values to estimate gets very large.

In Isaeva et al. [2011b,a] where MBR design was implemented if the number of parameters were larger than three, whilst running all combinations if there are less than four parameters. MBR design is also used for this implementation of DLU for mixtures and its five parameters. To keep the constraints for relationship between μ_i and σ_i some tricks were implemented after the design of runs. The design of runs assumes that all the other variables have the same values independent of the value in another variable. But here the σ_i will actually be dependent on μ_i, excluding some values from σ_i.

In this case, there are $k = 5$ design factors $\mathbf{w}_k$. These are mapped into indexing factors $\mathbf{d}_k$. The factors have different lengths, $L(k) = 2^{M(k)}$, but this is no problem for MBR. The extra conditions upon the σ_is create trouble for the MBR. The σ_i is dependent on the value of μ_i, and this is a limitation the parameter space. The parameter space for only two of the parameters, σ and μ_i, is reduced by half due to the condition that $\sigma_i < \mu_i/3$ and $\sigma_i < (1 - \mu_i)/3$. When half of the combinations for two of the parameters are removed, this affect all the other parameters. Every value of i.e. the mixing parameter ε is combined with every parameter value of σ_i and μ_i. This happens all over again with the second set of σ_i and μ_i parameters. The same restrictions apply for the second set of parameters, and will again reduce the parameter combinations by half. In total, the number of parameter combinations left will be a fourth of the original number of combinations.

A graphical illustration of the reduction in the parameter space is seen in figure 3.3. The planes depicted show the limits where the combinations are cut off. The combinations that would fit inside the triangle are still included, the combinations outside are not. The first horizontal axis are the values of a μ_i parameter, the vertical axis the values of σ_i and the last horizontal axis are a few points of the mixing parameter ε, to illustrate the shape of the figure.

It is difficult to illustrate how the combinations of parameters is reduced to one fourth of the total number of combinations when all five parameters

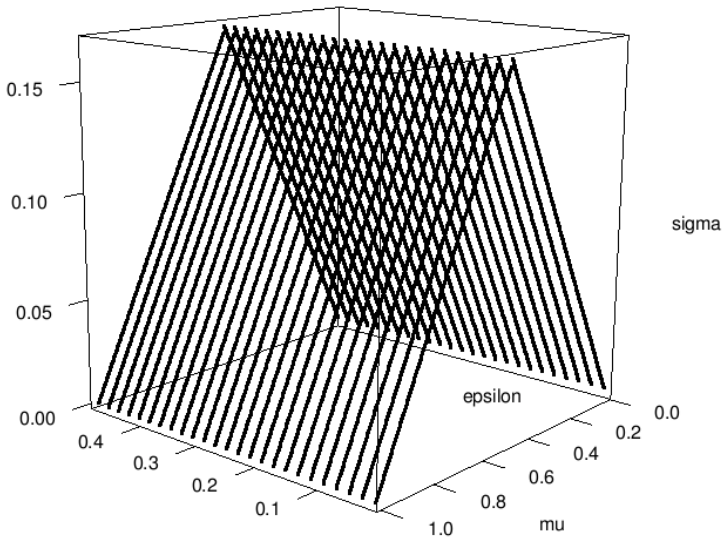


Figure 3.3: Illustration to demonstrate the reduced parameter space

are included, illustrations in five dimensions are difficult to put on paper.

The reduction in the parameter combinations means that for a large design, containing a large number of runs, only about a fourth of these will be run. This makes optimizing the database based on the design complicated, since the a lot of the traditional methods to optimize designs use the design matrix, which is reduced in the case of DLU.

Table 3.1: Parameters and number of levels in the parameter, i.e. length

k	M(k)	L(k)
ε	8	256
μ_1	9	512
μ_2	9	512
σ_1	7	128
σ_2	7	128

The decided lengths and levels of specificity chosen for the different

parameters are as described in table 3.1. If a full factorial design was to be run here, mapping the levels in to the factors, the number of lines to be combinations is of 10^{12} in magnitude.

The recoded factors are mapped into bit factors $\mathbf{f}_{k,1}, \ldots, \mathbf{f}_{k,M(k)}$, and the resulting bit factor matrix F is shifted into the matrix G containing the design, as described in sections 2.6.

A fractional factorial design using MBR is found for the chosen parameters. Literature and software can provide optimal sets of generators and alias structures, but only up to a certain number of factors. This problem is of such magnitude that alternatives to previously tested generators have to be found. To be able to find a fractional design, generators have to be specified. Choosing the confounding patterns, the resolution and finding the alias structure of this design for this estimation problem is a large issue.

3.2.1 Generators

There are reasons to believe that the generators will have an impact on the estimators. The generators decide which linear combinations of the parameters are to be included in the library. Choosing the defining factors and resolutions in higher order systems can be difficult. A proposed testing strategy is to randomly generate sets of generators from different type of constructs for confounding patterns. This include varying the size of the generators and different confounding strategies while choosing bits in the generators. The rules for specifying good generators that apply to ordinary fractional factorial designs does not necessarily apply to the MBR situation, as the generators and confounding issues are on the bit factors and not on the design factors.

The confounding issues and the generators are highly related to each other, and testing will be done to see how the generators influence the confounding of the parameters and if this has an impact on the prediction error of look-up. If the solution room is not properly spanned by the design of runs it will be reflected in the curve estimates.

To perform tests of the different strategies for generators, random designs and generator sets will be drawn, for each set of generators the design will be created and a number of curves from random distributions will be tried using the look-up method. The average mean squares error is found each time for the estimates and compared to see if there are differences between the strategies. This will remove the chance of stating that there are differences where there are none, based on one particularly good (or bad) combination.

3.2.2 Alias structures and resolution designs

The generators make up the defining relations of the design. The defining relations decide which combinations of factors that will be equivalent to each other.

Examples: In a fractional factorial design of size 2^{7-2} the two required generators can be defined as:

$$F = ABC \text{ and } G = ADE$$

The defining relations are then

$$I = ABCF = ADEG = BCDEFG$$

The shortest "word" in the defining relations is of four characters, which means that this is a resolution IV design. The different levels of resolution have different impacts on the design.

- Resolution III: no main effects are aliased to each other, but they can be aliased with two factor and higher interactions.
- Resolution IV: main effects are not aliased to two factor interactions, but two factor interactions are aliased to each other
- Resolution V: no two factor interactions are aliased to each other.

Fractional factorial designs and resolutions are all described in Montgomery [2009, p.292-293]. Higher resolution means less aliasing in the effects that are most likely to be of importance. Only looking at the resolution is not always enough to decide between designs. The number of words in the defining relation that have the shortest length also matters, since it defines the number of factors that will alias with each other. For more on aliasing, resolutions and defining relations, see Montgomery [2009, chapter 8].

To find a set of generators giving little trouble with confounding patterns and large alias structure, choosing a set of generators of resolution V or higher is generally good. Resolution of the design and the alias structure of effects are important criteria for choosing generators and together with the prediction error for the model based on the design, a good strategy for finding generators can be found. Alias structure and resolution will only work to decide the spanning capabilities of the bit factors. There might still be bad confounding patterns or poor spanning of the design space, which is why prediction error is the important deciding factor. The alias structure will also change after limiting the parameter

space by the conditions for values of σ_i for both mixture components. Optimality criteria for DoE situations can be performed on the information matrix M of the design matrix G.

$$M = G'G$$

There are different criteria, and they optimize different aspects of the information matrix. The A, D and E criteria all optimize the design matrix in one way or another. The optimality criteria uses the information matrix to make decisions about the design matrix. It is done by finding the determinant, the eigenvalue or the inverse matrix, and are all different ways that generally lead to a good design. These criteria optimize the design matrix, and for MBR they optimize the design matrix of the bit factors.

The G and V optimality criteria both focus on the prediction error of the design instead of the design matrix itself. This will show whether or not the different bit designs are different for creating good prediction models in the design space.

Looking at both aliasing, resolution and the various optimizing criteria, some guidelines for choice of generators should be deduced. If there are a few designs that have good optimality criteria, good resolution and confounding patterns and yield good prediction accuracy, the best one can be set as a default set of generators and design, with the option of specifying another.

A general hypothesis test situation can be set up to test for differences in the generator structures. The number of aliasing factors, the amount of prediction error and the optimality criteria can all be used. Testing will in general be to look for differences between groups depending on different settings and rules for setting up generators, using the number of aliasing effects, the optimality scores or the prediction error as a response, as the variable to establish different groups for.

Literature such as presented by Thalberg [2011] show that there is little to be gained by searching for the optimal design, so this will not be given too much focus compared to the implementation and optimization of the look-up procedure.

3.3 Preprocessing

Standardization of the range and values of the curves is necessary to make the library general. It is assumed that the curves in the library are from a

function following the forms of probability functions, that a density function has equal start and end values and that a distribution function is increasing.

For the distributions in the curve library, these conditions are fulfilled at their creation. The parameter values chosen in 3.2 removes the need for preprocessing of the curve library. This differ from Isaeva et al. [2011b] in the need of processing. The original implementation of DLU use the library of curves to estimate mathematical problems of all sizes, while this version of DLU is to estimate a probability function.

3.4 f or F?

Using the density $f(x)$ or the distribution $F(x)$ to create the curve library is a key point to the development. Estimating the parameters using the probability density function or the probability distribution function can be different. Determining between different functions will be decided by comparing compression results in PCA, depending on which type of function can most easily be reduced to the least number of principal components. The number of principal components is important to the practical performance of the DLU.

As for investigating differences between the generator choices and designs a set of randomly drawn conditions will be compared between the use of the density function or the distribution function. As the key point to this method is fast estimation and small stores of precomputed data, the number of principal components needed to retain 99.9% of the variability in the data is the main deciding factor for choosing between $f(x)$ and $F(x)$.

If no significant difference in the compression results, the average error of parameter estimation will decide.

3.5 PCA

A compression of the lines is performed to reduce the storage space required to solve the problem. The complete curve library can be expressed as

$$Z = \bar{Z} + TV' + E$$

where X is the collection of preprocessed distribution curves, $\bar{Z}$ is the arithmetic mean of the columns of Z, T and V' are the selected scores and loadings for this data. The scores and loadings are found using SVD or PCA and E the residuals.

Performing PCA for the DLU curve library van be done using SVD. This splits the data matrix into 3 matrices, the right and left singular vectors and the singular values.

$$Z = UDV'$$

The scores are $UD = T$ and the loadings are found as V'.

By keeping only the few first principal components, only a few of the columns of T and V', the data set can be reduced drastically in size. For many situations, more than e.g. 99% of the data can be explained through a set of components, and this allows data reduction. It creates the residuals, as if only 99% of the variation in the data set is explained, 1% is not explained, and does not fit into $Z = TV'$, and thus make up E.

The residual terms and variances of E are found using

$$e_j = (z_j - \bar{Z}) - t_j V'$$

and

$$s_j^2 = \frac{e_j \cdot e_j'}{n}$$

The vector z_j is the library distribution curve and t_j is the score value for curve j. The length of x_j, n is the number of observation points.

Performing SVD of the matrix of function values equips us with enough scores and loadings to reconstruct the values to 99.9% of the variance in the original generated data explained. The scores and loadings can be used to find an estimate for the new lines in the score space, find the closest library distance curve and compare to the reconstructed lines in the original space.

Having found the mean square error of all the curves in the library, the difference between the original library curve and the reconstructed value for that curve, a baseline error within the reconstructed library can be found. Errors that appear in these curves is the level of error to expect for a matching curve during a look-up.

After finding the scores and loadings for the library and reducing the amount of data, the important data to save are now the scores, loadings, the arithmetic mean of Z and the limits for the library error. If the distribution curve library is fit for compression using scores and loadings, the number of data points to save will have been drastically reduced during this process, as described in section 2.8.

3.6 Performing look-up

When performing look-up, a set of observations are to be estimated using the DLU method, the following form of data is assumed. The goal is to model the observations as a function $F(x)$ where F in this case is the mixture of two normal distributions.

$$z_i^{obs}(x) = F(x; \mathbf{p}_i) + e_i(x) \tag{3.3}$$

Here the $e_i(x)$ the error and $z_i^{obs}(x)$ the new observations over a range x. Hence, the aim is to find the $F(x; \mathbf{p}_i)$ that best fit the observed $z_i^{obs}(x)$.

3.6.1 Preprocessing the observations

The new observations must satisfy the same conditions as the library curves. Observations data are transformed to curves using a kernel density estimate, and either proceeding as a density function or a distribution estimate found from the density estimate.

If the new input observations are observed on an interval other than $x \in [0.001, 1]$, processing for the x-dimension need to be done. The offset in the x-direction equals the minimum observed value of x^{obs}, $off_x = x_{min_0}$. The slope parameter is found using $sl_x = x_{max_0} - x_{min_0}$. The value x_{max_0} is the maximum value of x_{input} and x_{min_0} the minimum value of the same. The value x_{max} is the maximum value of the desired x-range, in this case 1, and x_{min} is the desired minimum value, in this case 0.001.

To achieve the 100 points in $[0.001, 1]$, the following transformation is used.

$$x = x_{min} + (x_{max} - x_{min}) \frac{x^{obs} - x_{min}^{obs}}{x_{max}^{obs} - x_{min}^{obs}}$$

This transformation of x is not of great importance for the actual look-up, as the closeness of lines is performed index by index, but it is of great importance when returning the parameters after estimation, to have parameters that relate to the original observations.

The look-up process is now performed by finding the closest library curve, in the score space and in the design space.

3.6.2 Curve fitting

New curves are estimated by the same method as in section 2.5. This is a non iterative method of curve fitting. The new observations are fitted to the library in the score space as

$$t_i = (z_i^{obs} - \bar{Z})V$$

with the residuals and error found as

$$e_i = (z_i^{obs} - \bar{X}) - t_i V' \quad \text{and} \quad s_{e_i}^2 = \frac{e_i \cdot e_i'}{K}$$

The error can be compared to the baseline errors found in the generation of the distribution curve library, and potential fits are found.

3.6.3 Finding best models and parameters

To find the best distribution and parameter set, the distance in score space is used. The distance between the new curve i and the library curves j is computed as:

$$s_{t_{i,j}} = \sqrt{(t_i - t_j)(t_i - t_j)'} \tag{3.4}$$

Choosing the curves with the least error and distance identifies the curves in the library that most closely resemble the new observation. Model parameters are then chosen as the model parameters of the best fits of the library curves.

The best curves chosen are expressed as $F(x, \hat{\mathbf{p}}_i)$, where $\hat{\mathbf{p}}_i$ is the parameter set of the i'th solution, $\mathbf{p_i}$. The 10 top fits are reported back as alternative solutions.

3.6.4 Removing the preprocessing

The initial preprocessing of the curves need to be removed before a solution is complete. To remove the preprocessing in the x-dimension, following procedures are made.

For μ_1 and μ_2 there are

$$\hat{\mu}_{input} = \frac{sl_x\hat{\mu} - x_{min}}{x_{max} - x_{min}} + off_x$$

and for σ_1 and σ_2 there are

$$\hat{\sigma}_{input} = \frac{sl_x\hat{\sigma}}{x_{max} - x_{min}}$$

This shifts the expectation and variance of the distributions back to the original input x dimension. The values x_{min} and x_{max} refer to the minimum

and maximum values of the x-range in the curve library, 0.001 and 1. The offset off_x and slope sl_x parameters are used to transform the x-range of the observation values to the desired x-range for the library look-up.

These preprocessing and post-processing factors are safe to perform on data, as if they are functions on $x \in [0.001, 1]$ it will not change the range or the parameter estimates, and if x is on a different interval, the post-processing will allow the parameter estimates to describe the actual values related to the sample values of the new observations.

The last parameter of the mixture model problem, ε, is not changed in processing, as this is only the relative weight of one distribution compared to the other one.

As the estimation result for a probability distribution is the estimated values for the parameters and the values in the y-dimension used for standardising the density estimate to the curve library no post-processing is needed for the y-dimension.

3.7 Measure the performance of the model

To measure the performance of the model, estimations of the error of the model is made. The prediction error and log likelihood values for the model can be used to measure the performance.

The performance of the model can be described using the error of prediction and the error of parameter estimation. The prediction error is based upon the difference between the predicted values and the true known values. This difference describes the accuracy of the prediction.

The log likelihood estimate is a measurement of how well the estimated function fits the observed data.

Comparisons with established estimation methods is an important part of the development of a new method. This is to decide whether or not the new method is of use, or if the old methods work better for now.

3.7.1 Prediction error

The prediction error can be used to evaluate the performance of the DLU. Taking the mean square error of prediction for each line in the model allows to find a general measure of how well the model fit to this exact line. If the prediction error is within the error limits established while running the library, it can be claimed as a probable fit to the library curves. This prediction error is an error estimate that will decrease as the model complexity increases, and is therefore not a good estimator of generalized

error. It can however be used on models within the same size of complexity to give an indication of which works better, by comparing this relative error.

Running different types of generator designs can influence the estimates, and change the prediction error. The different design also report different curves selected as the best fits. Different generator designs can be compared in terms of MSEP of new lines.

More specifically

$$\text{MSEP}(\mathbf{x_i}) = \frac{1}{n}(E(\mathbf{Y} - \hat{f}(\mathbf{x_i}))^2)$$

Comparisons on this level compare the error of prediction between new curve estimates. Averaging over all curves in the library of distribution curves results in the final error of prediction for that choice of generator. Generator optimizing criteria that use the prediction error will seek to minimize this average, find the maximum error or other criteria based on the prediction error.

This error can be used to find out if there are differences between types of generator sets, design specification and sample observations.

The prediction error is on the form

$$E[\sum_{i=1}^{n}(g(x_i) - \hat{g}(x_i)^2)]$$

The traditional estimates for prediction error and estimation error can be produced using boot-strap or cross-validation methods, and the lack of these estimates is one of the weaknesses mentioned in Isaeva et al. [2011b], as many statisticians will want these estimates. However, there are other error estimates that will provide similar information.

Estimation of the prediction error for the methods EM and MCMC also use methods like boot-strap and cross-validation.

3.7.2 Log likelihood comparison

The estimation problem in parametric estimation of a mixture distribution can be expressed as a likelihood function. To estimate a test error, the log likelihoods can be found for an independent test set. This is an estimate of the test error using the log likelihood as a loss function, and it is well adapted to this type of model estimation. Choosing parameter values and model based on likelihood functions is a form of maximum likelihood estimation. This is expressed as in equation 2.23. Parameters are not chosen based on log likelihood value, but the models are evaluated using log likelihood values to compare the methods.

The goal of a maximum likelihood estimator is to find the parameter set that will maximize the likelihood of the data. The likelihood can be used as a loss function to give an estimation of the test error and give a means of comparing methods. A common way of comparing likelihoods is through ratios. A ratio on a log likelihood will the be a difference between the logs, since $log(\frac{l_1(x)}{l_2(x)}) = log(l_1(x)) - log(l_2(x))$. The likelihood ratios, or the difference in the log values, can be compared between all models of equal size. The mixture model is a model of five parameters for all estimation methods, and the log likelihood values can be compared directly.

The maximized log likelihoods can also be used to see if there are any differences in the different sizes of curve libraries and generator size for the DLU, to see whether one type of library give better results than others. For comparisons in log likelihood values, it is the number of parameters that decide the complexity.

Log likelihood values for the predicted curve is chosen to compare sizes of the curve library and general performance of DLU to EM and MCMC in the parametric estimation of mixture models. This measure of prediction error or parameter validation can be performed over an independent test set to achieve a better estimate of a generalized prediction error than the training error found while selecting the best fits.

The log likelihood values "loglik" are given by:

$$loglik = \sum_{alli} log(\hat{\varepsilon} f(x_i; \hat{\mu_1}, \hat{\sigma_1}^2) + (1 - \hat{\varepsilon}) f(x_i; \hat{\mu_2}, \hat{\sigma_2}^2)) \tag{3.5}$$

A measure of the prediction error is then given as:

$$Err = loglik_{\hat{\boldsymbol{\theta}}} - loglik_{\boldsymbol{\theta}} \tag{3.6}$$

The $loglik_{\hat{\boldsymbol{\theta}}}$ is the log likelihood of test data for the estimated parameter, while $loglik_{\boldsymbol{\theta}}$ is the log likelihood of the test data for the true parameters behind the observations and test data.

This difference is a relative measure of difference between the estimated parameters and the original parameters, and can be compared over models of different sizes and complexity. Averaging over this relative difference for both EM, MCMC and DLU provides a measure to compare for model accuracy of the three methods.

3.8 Implementation

The actual implementation of the DLU for mixture models has been performed using R and a variety of methods and packages standard and external. Code and functions that have been used to create the curve libraries and MBR designs are included in the appendix B.

3.8.1 Established methods

EM and MCMC are the methods chosen for comparison with the DLU method. They have different approaches to the same estimation problem. As all of MCMC, EM and DLU are estimating models of the same size, the log likelihood values can be compared directly. The estimated parameter values can be compared to each other and the true parameters. Finding the prediction errors and estimation errors are a more complicated process for these methods than finding the log likelihood values.

MCMC

Specifically in the case of implementing MCMC in DLU for mixture models, the data $\mathbf{D}$ is made up by $\{\mathbf{x}, \mathbf{z}\}$ where $\mathbf{x}$ is the observed data points and $\mathbf{z}$ the unknown variable defining which component the elements of $\mathbf{x}$ is part of.

To implement and run the MCMC model, the opensource software OpenBUGS, the following priors are chosen as part of the MCMC model:

$$
\begin{aligned}
\mu_1 &\sim N(0, \frac{1}{10^{-6}}) \\
\mu_2 &= \mu_1 + \theta \\
\theta &\sim N(0, 1000) \\
\sigma_j^2 &= \frac{1}{\tau_j} \\
\tau_j &\sim G(0.001, 0.001) \\
\boldsymbol{\varepsilon} = \{\varepsilon_1, \varepsilon_2\} &\sim D(\mathbf{d}) \\
z_i &\in [1 \dots k] \text{ for } k = 2 \\
p(z_i = 1) &= \varepsilon_1 \\
p(z_i = 2) &= \varepsilon_2 = 1 - \varepsilon_1
\end{aligned}
$$

The priors are by choice vague, to avoid influencing the method too much by a possibly poor choice of priors.

After the implementation of the model, choice and generation of the priors, the model is run using R2OpenBUGS script in R, see Sturtz et al. [2005], and the OpenBUGS software. The OpenBUGS software is a tool specifically for Bayesian analysis of statistical models using MCMC.

This software allows the results to be gathered and analysed in R. Models are written and programs run from R, allowing OpenBUGS to perform the analysis and return the results back to the R interface again. Log likelihood values are found for the results and can be compared to the log likelihood values of EM and DLU.

EM

For the purposes of this thesis, the implementation of EM found in the R package mixtools is used. See package details Benaglia et al. [2009]. This package includes functions specifically written to find the two component mixture of sample data, and is written to speed up things for this particular problem.

Data from a mixture model is simulated and initial guesses of the parameters are computed based on the data. Initial values of μ_1 and μ_2 are the 25% and 75% quantiles of the simulated or observed data, while the initial values of σ_1 and σ_2 are the sample standard deviation for each of the halves of the sample data. The mixing parameters starting value will usually work well at 0.5, as that assumes equal weighting for both components. These are just the initial guesses that start the algorithms iterations.

The EM algorithm returns the estimated parameter values, the number of iterations it needed to stabilize and the log likelihood values for all steps in the algorithm. These are used to assess how well it found the solutions, and how much time it took to find better solutions with a higher log likelihood value. The log likelihood values can easily be compared to those of the other methods to find how the methods perform compared to each other.

3.8.2 Time

System time can be found using R's built in function for reporting time, see the system.time() in the base package R Development Core Team [2011]. It measures CPU time and user input time. When performing estimation for observations sampled from a mixture of random distributions the system time is recorded. Measuring the system time for a estimation procedure is highly dependent on the computer system where estimation is performed.

All times are given in seconds and parts of seconds. The function system.time() from R base is used to find the times of estimation. The system time function returns several parts, "user time", "cpu time", "elapsed time" and the user and CPU time of child processes. On Unix based operating systems times are reported with higher accuracies than on a Windows system, and the Windows systems are not able to report the times of any child processes.

For estimation using the look-up method time including finding all closest lines, estimated parameter values and errors and distances is recorded. MCMC in this implementation depends on external processes, and system times of the R process and the external processes is measured. However, a direct implementation of MCMC in R could increase the speed of this method. Performance time of EM is recorded when estimating parameters.

Chapter 4

Results

During the development and exploration of a new method, extensive testing is performed. New questions arise through development, revealing new problems and new needs to address. Results from these questions and the testing of the method is presented here, where the issues of development are presented first, then the results of comparing the performance of the DLU method to other established methods.

4.1 Compression

Compression using PCA is used as a tool for data reduction, and in the case of DLU to reduce the physical size of the curve library. As the function to include in the library is either the density function $f(x)$ or the distribution function $F(x)$, compression is performed for both alternatives, to find possible differences in compression results.

Another question to consider is whether to center data or use the original data in the compression and for the look-up function. Centering the data before analysis and compression is done by removing the column average from each column, and will remove any overall trend or characteristic expressed by the average values.

Score plots, scree plots and loading plots are diagnostic tools used to verify the compression into principal components. A score plot is a scatter plot of two principal components at a time. The scores of the three first principal components are reported from a curve database generated for a set of random generators. The design used in the plots is a fraction 25 design, to reduce the size of the plots by reducing the number of points in the library. The general shapes, percentages and figures stay the same for all generator sets and confounding patterns that have been tried. The

figures 4.1 through 4.4 shows the scatter plot matrices of the first three principal components. The odd shapes reflect the parameter restrictions and the design of experiments that have generated these data.

A loading plot is a lineplot of the values of the loadings. The plots seen in figures 4.6 and 4.5 presented here include the first four of the loadings.

The scree plots as seen in figures 4.7 through 4.10 are a good visual aid in seeing how much each principal component adds to the cumulative proportion of variability in the original data.

Scoreplots, loadingplots and screeplots are showed for one curve library, as examples of compression results.

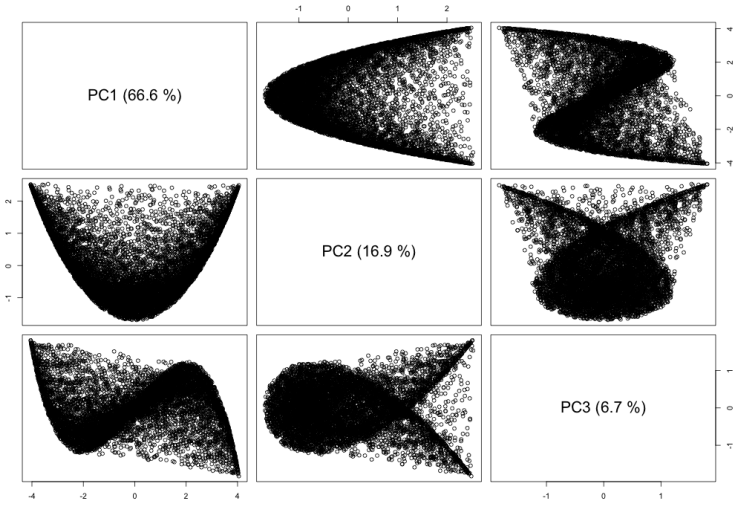


Figure 4.1: Scoreplot of the three first principal components of PCA on centred data from the distribution function $F(x)$.

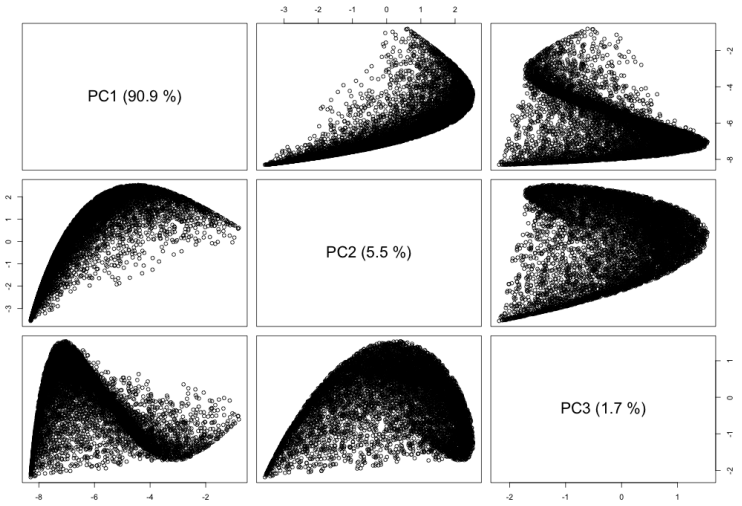


Figure 4.2: Scoreplot of the three first principal components of PCA on uncentred data from the distribution function $F(x)$.

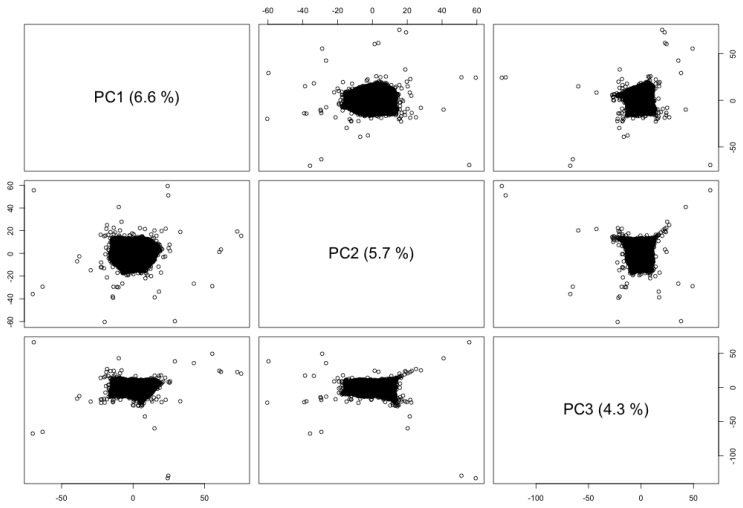


Figure 4.3: Scoreplot of the three first principal components of PCA on centred data from the density function $f(x)$.

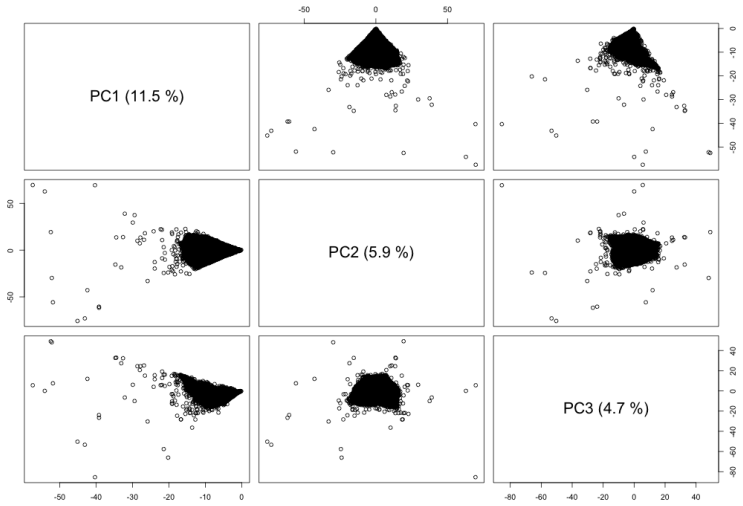


Figure 4.4: Scoreplot of the three first principal components of PCA on uncentred data from the density function $f(x)$.

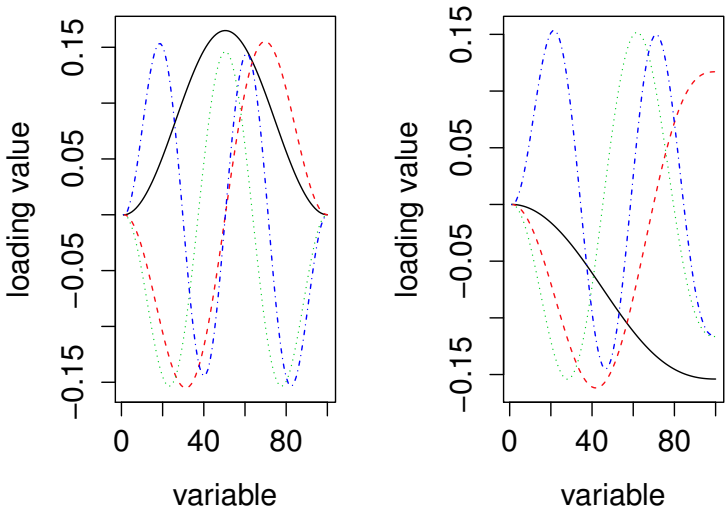


Figure 4.5: Loadingplot for the distribution $F(x)$. Centred analysis in the figure to the left, uncentred to the right.

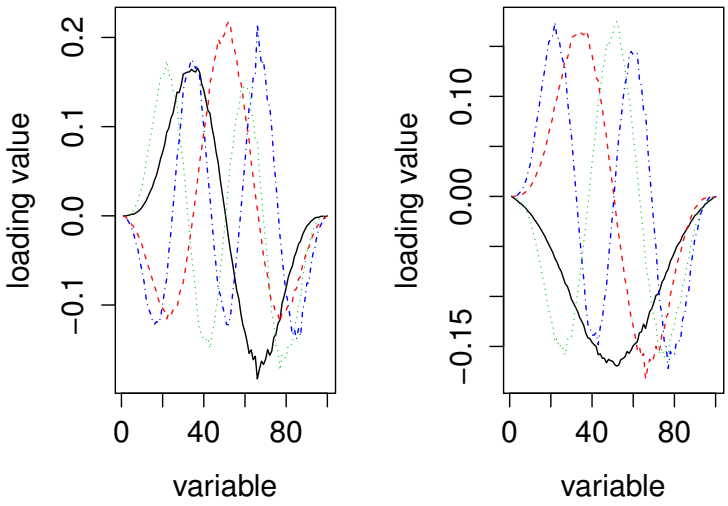


Figure 4.6: Loadingplot for the density $f(x)$. Centred analysis in the figure to the left, uncentred to the right.

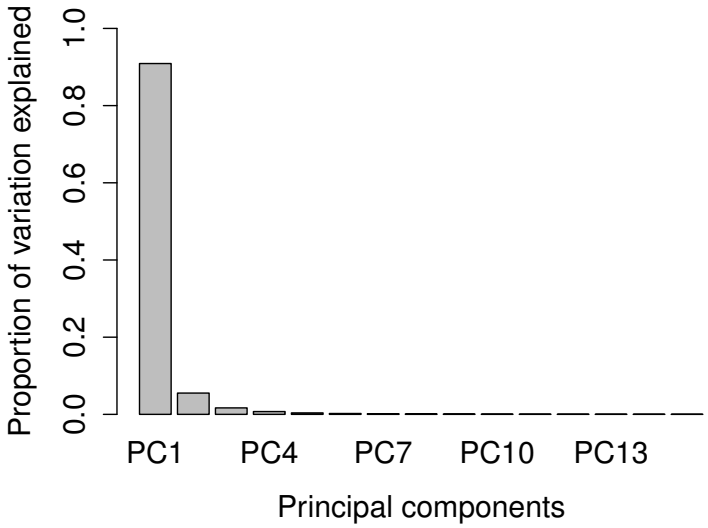


Figure 4.7: Screeplot of F(x), displaying how much variance can be explained by each of the first 15 components by compressing original data for $F(x)$.

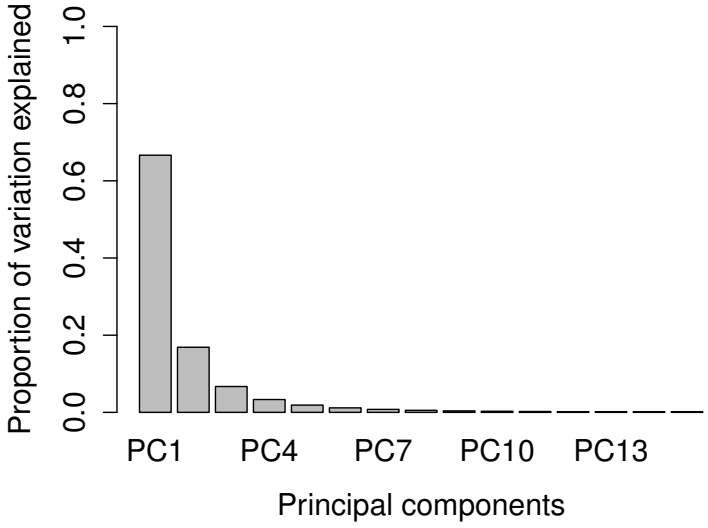


Figure 4.8: Screeplot of F(x), displaying how much variance can be explained by each of the first 15 components by compressing centred data $F(x)$.

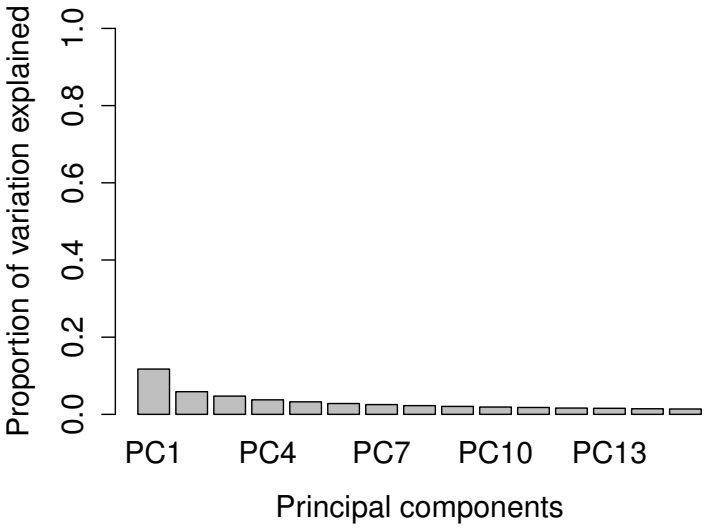


Figure 4.9: Screeplot of f(x), displaying how much variance can be explained by each of the first 15 components by compressing f(x).

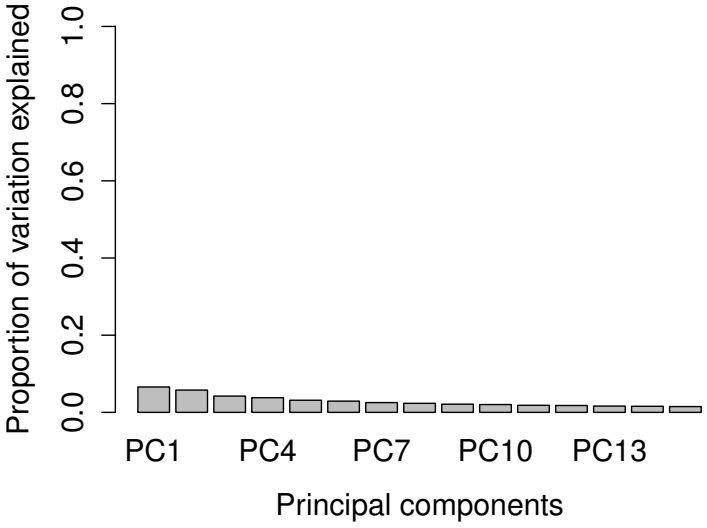


Figure 4.10: Screeplot of f(x), displaying how much variance can be explained by each of the first 15 components by compressing centred data f(x).

4.1.1 f(x) or F(x)?

The first decision to be made is between the probability functions, which of the density function $f(x)$ or the distribution function $F(x)$ would be preferred. Testing between the density function $f(x)$ or the distribution function $F(x)$ for the implementation of DLU the PCA was performed for both functions. The number of principal components needed to explain 99.9% of the variability in the data were recorded.

Drawing random generators following different rules of generators allow different confounding patterns and results. The number of principal components needed in each case to capture 99.9% of the variability in the data are recorded in table 4.1. These results represent the compression results for different sets of generators for different sizes of the curve library. Explanations of the different generators are found in 4.2. The variation is larger for the various full and odd confounding patterns, and less for the various other sets of generators. The overall pattern is still clear, in testing for differences for $f(x)$ or $F(x)$ and for centred versus uncentred data. The differences between these groups are much larger than the small differences within the groups.

The difference between the principal components of the density and the distribution is apparent from the scree plots as well as the number of principal components needed. Using a scree plot such as in figures 4.7 and 4.9 gives a visual to how much the data can be explained by the first principal components.

The scree plots in 4.7 and 4.9 shows the amount of variation explained using uncentred data. If the data are centred before running the principal component analysis the amount of explained variations look different. In this case the column mean of all the columns in the data is subtracted, taking with it a lot of the variance. The figures 4.8 and 4.10 display the same example of design experiment, using PCA on centred data instead of uncentred data. All four of the screeplots show the same number of components, despite the difference in how much variation is explained by these 15 first components.

The library generated with distribution function $F(x)$ as the probability function is dependent on the design and the levels of parameter values. An example of the curve library with the distribution function as probability function in seen in 4.11. These lines are examples from the curve library, 60 curves spread throughout the library.

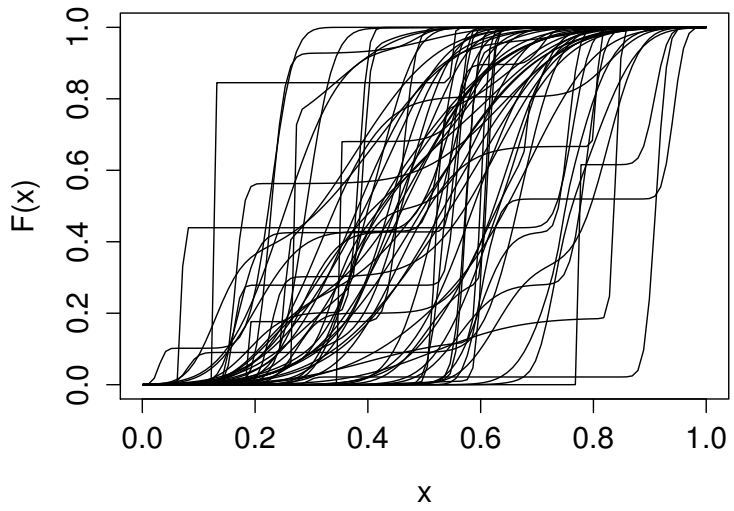


Figure 4.11: Example showing 60 random curves from a curve library of values for the distribution function $F(x)$. Displayed curves are from throughout the library.

Table 4.1: This table shows how many components are necessary to reach 99.9% of the variation explained for each of the density function $f(x)$ and the distribution function $F(x)$.

Groups		Centred		Uncentred	
Fraction	Name	f(x)	F(x)	f(x)	F(x)
22	6	98	39	98	16
22	7	98	39	98	16
22	8	98	39	98	16
23	6	97	39	97	16
23	7	98	39	98	16
23	8	98	39	98	16
24	6	97	39	97	16
24	6	98	39	98	16
24	6	97	38	97	16
24	7	98	39	98	16
24	7	98	39	98	16
24	7	98	39	98	16
24	8	98	39	98	16
24	8	97	39	97	16
24	8	97	38	97	16
25	7	97	38	97	16
25	7	94	33	94	14
25	7	98	39	98	16
24	Full	79	28	78	15
24	Full	87	35	87	21
24	Full	78	28	77	15
24	Odd	89	33	89	15
24	Odd	90	33	89	16
24	Odd	88	32	87	15

4.2 Generators

The number of parameter combinations is too large to run a full factorial design, and a fractional MBR design were found to reduce the size. The generators and confounding patterns was specified. It is believed that the generators will influence the prediction error of the look up results.

Traditional generator patterns such as full-confounding and odd-confounding were tried, and experimenting to reduce the number of alias effects, sacrificing resolution, by varying the bit size of generators. A full confounding pattern is found by including all the highest interaction effects in the generators, and an odd confounding pattern is a pattern of the highest interaction effects including an odd number of factors in the interaction. Both of these include randomly selected factor combinations at the lowest level, as it will often have too many unique combinations to include them all.

For a fraction 24 design, length 2^{40-24}, 24 generators were specified as combinations of the 16 first bit variables. A full confounding pattern requires that the combination of all 16 and every of the 16 combinations of 15 of the factors are needed. Lastly, 7 more generators are needed, and they were drawn randomly from the 120 different possibilities for combining 14 of the 16 factors. For the odd confounding, the factors that were included are the 16 combinations of 15 of the factors, and 8 randomly drawn generators from the 560 possible combinations of 13 bits out of the 16.

The random generators chosen with a set number of bits were found. The idea behind these are to have shorter defining relations than found for the odd and full confounding patterns. There is a possibility that the randomly drawn generators will influence the design.

As optimizing criteria that work on design alone is not of much interest for the effect of how well the design works with MBR design, optimizing criteria that work on the prediction error is a better approach. The generators are evaluated in design space, and not in the bit space.

As the goal of selecting generator sets are to achieve good predictions and parameter estimation, the natural selection criteria becomes the prediction errors of look-up. The prediction error can be used to compare models of the same size of the curve library. A large curve library will in general give a smaller prediction error than a smaller curve library, as more of the parameter combinations are included, creating a more accurate library. These prediction errors are in sample estimates of the error, as they are the errors of the chosen fits against the input values to the look-up method.

The log likelihoods of the estimated parameter sets were compared to

find differences in whether or not library sizes have an impact on the final predictions. As the different sizes of libraries don't change the number of parameters the log likelihood values can be compared directly, also for different library sizes.

4.2.1 Prediction error

The average prediction error over a large set of observations can be used for model selection within the same complexity. Prediction errors of generator sets for designs of the same fraction and mode can be compared.

The set of observations used as a training set for model fitting and error estimation is a set of 800 different observations sets. All parameters are randomly generated within the same range. The sample sizes are 50, 100, 200 and 500, and there are 200 samples for each of these sample sizes. DLU starts with a numerical estimate of these to a density estimate of 100 points.

The look-up error is recorded over a large set of observations, estimated using 5 different groups of libraries, 3 of each group. All five groups contain designs of 2^{40-25} designs. The groups and the average error in each of these is recorded in table 4.2. Analyse of variance tests were performed for each of the groups described below, to find whether fraction 24 design with 8 bit generators differ from i.e fraction 24 design with 6 bit generators, or generators from an odd confounding pattern. The groups containing the odd and full confounding patterns have higher error averages than the first three groups described.

From the table 4.2 it is seen that the average prediction error of the first 9 means are smaller than for the last 6. It seems that odd and full confounding patterns give higher prediction error in a library of the same size. There are significant differences in the prediction error, see the table 4.3. Finding where these differences are can be done with the contrasts technique. Both odd and full confounding patterns are different from the random generators of set lengths. The significant probability values are 0.0014 and 2×10^{-12} for odd and full confounding patterns, respectively. There are no differences between the groups of random generators of set lengths, contrasts in the analysis yield probabilities of 0.66, 0.80 and 0.49 for groups being equal when comparing each of the groups to the others. Prediction errors are lower for random generators of set lengths than for the odd and full confounding patterns.

For the generators that have been randomly generated, it seemed that the

Table 4.2: The mean prediction error for each of the groups involved in the analysis of varaince for the prediction error

Group	nr	means
8	1	0.00023901
8	2	0.00023905
8	3	0.00026482
7	1	0.00025167
7	2	0.00023680
7	3	0.00023443
6	1	0.00024290
6	2	0.00023905
6	3	0.00023692
Odd	1	0.00028253
Odd	2	0.00026986
Odd	3	0.00027095
Full	1	0.00028597
Full	2	0.00037485
Full	3	0.00027621

Table 4.3: Full analysis of variance comparing the prediction error in a MBR library of length 2^{40-24}, for random generators of decided lengths and odd and full confounding patterns

Anova	Df	Sum Sq	Mean Sq	F value	Pr(>F)
Groups	4	$9.2024e-06$	$2.3006e-06$	13.0389	$1.3487e-10$
Residuals	11995	$2.1164e-03$	$1.7644e-07$		

third of the generator sets of generator length of 7 bits is the one with the lowest average prediction error, and was used for further examples of the look-up. Selecting the design with minimum prediction error is according to optimality criteria, specifically V-optimality which seeks to minimize the prediction error.

4.2.2 Log likelihood

The log likelihood values of the parameter estimates over a large number of lines can be used to assess the difference in generators for libraries of different sizes. This log likelihood value depends on the size of the estimated model in the number of parameters involved, and this number is constant.

The log likelihood values were compared to the log likelihood of the true parameters on an independent test set, a test set with the same parameters, but a different sample. All differences were computed, as described in eq. 3.6.

The average difference in log likelihood values was also found for the same structure as for the prediction errors, and is seen in table 4.4. The parameter sets were found for 800 sets of observations, run for every library, and the log likelihood values were found for a corresponding test set. There is one test set for each set of observations, with the same true parameters. The differences were averaged in each group. The analysis of variance results for difference in the log likelihood values are shown in the table 4.5 .

Extending the log likelihood experiment to include examples of libraries of other sizes as seen in table 4.6, it was found that there was no difference in the log likelihood values between library sizes either (see table 4.7). The means were however higher for larger sizes of the curve libraries here as well. The higher the value, the better the fit, but values can not be compared in general. The likelihood values of a mixture with expectation and variance parameters in the range of i.e. $\{0; 1\}$ will be very different from the likelihood values of the same mixture, where the expectation and variance parameters range from i.e. $\{-10; 10\}$

The log likelihood estimates here were all computed over the same set of observations. All the observations were generated on the same range of random parameters, as were the test sets. This was to better find differences between the libraries, whether the different generator sets provide different solutions or not. If the generators did provide different solutions, there would be different means for the different generator groups.

The group described as “dense” is of the same library size as those of fraction 24, but the parameters were much denser, there were more generators and it was conducted as an experiment to see how it would

Table 4.4: Average log likelihood values for the parameter estimation performed in a 2^{40-24} design with group names.

Group	nr	Average difference
8	1	−41.1576
8	2	−75.5737
8	3	−33.0088
7	1	−42.2301
7	2	−40.0084
7	3	−36.0479
6	1	−37.5933
6	2	−75.5737
6	3	−49.4599
Odd	1	−47.0982
Odd	2	−36.6902
Odd	3	−50.0625
Full	1	−45.9803
Full	2	−49.3160
Full	3	−48.5955

Table 4.5: Table showing analysis of variance of the log likelihood values for parameter estimation

Anova	Df	Sum Sq	Mean Sq	F value	Pr(>F)
Groups	4	298326.1502	74581.5376	0.8404	0.4993
Residuals	11977	1062925497.5997	88747.2236		

compare to the other groups. The smaller library size, of fraction 25, is a similar experiment. This group contained half the number of curves as compared to a fraction 24 design, but without much change in the log likelihood values. There was one group consisting of the larger fraction 23 design, of various bit sizes, and one with the even larger fraction 22 design.

The desired log likelihood differences are to be as close to 0 as possible. If it is negative, its fit is worse than the true parameter, and if it is positive, the fit is better. Positive log likelihood differences indicate either a very skewed observation sample that fit badly with the true parameters and that the estimated parameters are better, or that the model is overfitted to the observation data. This risk should be less when finding the log likelihood values over an independent test set, and reduce the number of instances when the estimated parameters are a better fit than the true parameters.

Table 4.6: Average difference in log likelihood, groups and characteristics of the means.

Group	Name	fractions	nr	Average difference
1	8	24	1	−41.1576
1	8	24	2	−75.5737
1	8	24	3	−33.0088
2	7	24	1	−42.2301
2	7	24	2	−40.0084
2	7	24	3	−36.0479
3	6	24	1	−37.5933
3	6	24	2	−75.5737
3	6	24	3	−49.4599
4	6	22	1	−34.1419
4	7	22	1	−40.3553
4	8	22	1	−39.4921
5	6	23	1	−34.5287
5	7	23	1	−38.5290
5	8	23	1	−31.6625
6a	Dense	24	1	−37.6248
6b	7	25	1	−44.8878
7	Odd	24	1	−47.0982
7	Odd	24	2	−36.6902
7	Odd	24	3	−50.0625
8	Full	24	1	−45.9803
8	Full	24	2	−49.3160
8	Full	24	3	−48.5955

Table 4.7: This anova table show the analysis of variance for an extended set of groups.

Anova	Df	Sum Sq	Mean Sq	F value	Pr(>F)
Groups	8	740930.3736	92616.2967	1.3723	0.2030
Residuals	18363	1239303850.8690	67489.1821		

4.3 Performance

Performance of the chosen model is the most important result for implementation, use and analysing the DLU method. When testing performance of a new method, both accuracy and speed is of importance. How well a method works is of most interest when compared to how other methods perform.

Performance was tested over the same generated distribution data as used for determining the better MBR designs and generators. For methods like EM and MCMC, sample data is used for input to the estimation algorithm. For DLU a probability function curve is approximated before look-up in a previously defined database. This preprocessing allows the transformation from a random data sample to a an approximated distribution curve, where the goal is to establish the parameters using DLU.

4.3.1 DLU

Prediction accuracy for generators that work is measured by the RMSEP. This was found using the errors between the predicted curves and the new observation. The error was used to define if a curve is a fit to the input observations and evaluate the fit. To demonstrate a return of the look-up experiment two examples are given here with two more in the appendix. The ones shown here are estimation situations where there is a clear mixture, while the two in the appendix are where the mixing parameter is very small, so that the situation is almost a single normal distribution.

Looking at the parameter values of table 4.8 or 4.9 it is difficult to get a picture of exactly how the differences in parameter values impact on the mixture distribution itself. Taking the function parameters as input to a density function on the x-range of the input parameter, plots can be produced to have a visual guide to how well the parameters fit. The plot 4.12 contains the plot of all the look-up solution as described in 4.9. The red line is the density function on by the true parameters and the red stapled line is the numerical density estimate of the sample observations. Examples of plots comparing the estimation methods are presented later in figures 4.14 and 4.15. These contain the three first estimated parameter sets of the DLU results together with the parameter estimations of EM and MCMC.

4.3.2 EM and MCMC

EM and MCMC are chosen as the methods to which DLU is compared. Both methods report the number of iterations needed to reach a final

Table 4.8: The full table of DLU look-up results including errors and distances for the returned ten curves of a small sample size

	μ_1	μ_2	σ_1	σ_2	ε	Error	Distance
True	1.7964	−1.5660	0.7599	1.6950	0.3988		
1	2.7320	−1.0746	0.5228	1.9953	0.0628	$2e-04$	0.1384
2	1.8680	−1.3715	0.6502	1.8862	0.1745	$3e-04$	0.1647
3	1.7600	−1.5336	0.7228	1.9589	0.1372	$3e-04$	0.1706
4	0.0593	−1.5875	1.6499	2.2861	0.3647	$3e-04$	0.1753
5	1.2741	−1.1555	0.9228	2.1225	0.1274	$3e-04$	0.1854
6	1.5711	−0.9666	0.7047	2.0679	0.0216	$3e-04$	0.1870
7	−2.3164	−0.6426	2.0498	2.0134	0.1588	$4e-04$	0.1876
8	−2.9644	0.1673	1.3227	1.6318	0.3882	$4e-04$	0.1939
9	−6.5009	−0.9126	0.1412	2.0498	0.0216	$4e-04$	0.1953
10	2.7590	−1.2096	0.4320	2.0862	0.0314	$4e-04$	0.2050

Table 4.9: The full table of DLU look-up results including errors and distances for the returned ten curves of a large sample size

	μ_1	μ_2	σ_1	σ_2	ε	Error	Distance
True	−0.6308	−0.2483	0.3930	0.9211	0.4713		
1	0.8437	−0.5512	0.3723	0.6320	0.0745	$1e-04$	0.1223
2	−0.0863	−0.6987	0.7465	0.5480	0.4353	$2e-04$	0.1364
3	1.2747	−0.5966	0.4946	0.5327	0.1000	$2e-04$	0.1506
4	0.0385	−0.6533	0.7465	0.5862	0.2608	$2e-04$	0.1548
5	−0.8688	−0.1996	0.5327	0.7542	0.3608	$3e-04$	0.1595
6	−0.6419	−0.3357	0.2884	0.8763	0.4039	$3e-04$	0.1597
7	−0.0522	−0.6873	0.9145	0.3571	0.4902	$3e-04$	0.1600
8	0.4808	−0.6987	0.7618	0.6167	0.2294	$3e-04$	0.1613
9	0.4581	−0.5966	0.5022	0.6243	0.1412	$3e-04$	0.1614
10	−0.9141	−0.3584	0.3113	0.7160	0.1412	$3e-04$	0.1655

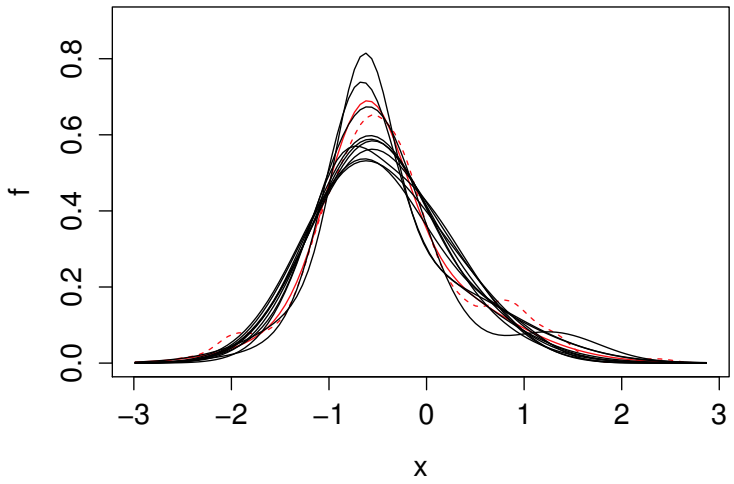


Figure 4.12: Plot of $f(x)$. True parameters in red, the non-parametric density estimate in stapled red, and the DLU estimates in black. Sample size: 500

estimate, and are both limited to a specific number of iterations before forced to stop. The number of iterations depend on the observations to be estimated. MCMC is limited to 3000 iterations, where the first 1000 are removed as a burn-in period, while for EM it is a 1000 iteration maximum before stopping the iterations. If convergence is achieved, the iterations stop.

4.3.3 Comparison

Estimation results

Comparing the estimation results for the methods the DLU estimations are compared to parameter estimations of EM and MCMC. The 800 sets of observations used to investigate differences in generators and designs are used to test the chosen design as well.

Examples of parameter estimation using the look-up method and the results that were reported back are presented here. The first situation is of a small sample size, and parameters for the sample of data as seen in the top table of 4.10. The second example estimated from the observations of the parameter values as in the top table of 4.11 are of a large sample size. The true values are the parameter values behind the sampled observations.

The sample values are calculated when the sample is generated, based on knowing which distribution the samples are from, and not from the parameters. In the following pages the results of a DLU look-up is presented. For two more examples of very different parameter values of the true parameters, see the appendix A.2.

Looking at the figure 4.13 the distribution functions of the estimated parameters of the two different examples are displayed. The figures 4.14 and 4.15 show the density function of the same estimated parameters. These plots were used to illustrate the differences between the true distributions, the estimation methods and the non parametric estimate of the sample. Using the difference in log likelihood values as a measure of fit, some summarized data can be seen in table 4.12. Here the minimum, maximum and average values for the log likelihood values are reported, along with the variance. The summary statistics here are found over separate columns, allowing comparison between the EM, MCMC and DLU methods.

These values are differences from the log likelihood of the true values, and are an estimate of test error, as the log likelihoods are found over an independent test set, simulated on the same parameters as the observation data.

The bottom tables of 4.10 and 4.11 could also include the seven next of the returned DLU lines, and as seen they vary. It is not consistent that the

Table 4.10: Example observations, sample size: 50. Top: True parameters of simulation. Bottom: Comparison of parameter estimates and the differences in log likelihood values.

	μ_1	μ_2	σ_1	σ_2	ε	$1-\varepsilon$
True values	1.7964	−1.5660	0.7599	1.6950	0.3988	0.6012
Samples	1.8433	−1.4792	2.6268	2.3381	0.3200	0.6800

	True	EM	MCMC	DLU1	DLU2	DLU3
μ_1	1.7964	1.8084	0.5157	2.7320	1.8680	1.7600
μ_2	−1.5660	−1.7345	−1.2081	−1.0746	−1.3715	−1.5336
σ_1	0.7599	0.6922	1.8517	0.5228	0.6502	0.7228
σ_2	1.6950	1.6232	0.9277	1.9953	1.8862	1.9589
ε	0.3988	0.2456	0.4888	0.0628	0.1745	0.1372
loglik	−106.80	−106.30	−115.65	−107.84	−106.23	−107.17
diff	0.0000	0.4975	−8.8508	−1.0406	0.5696	−0.3763

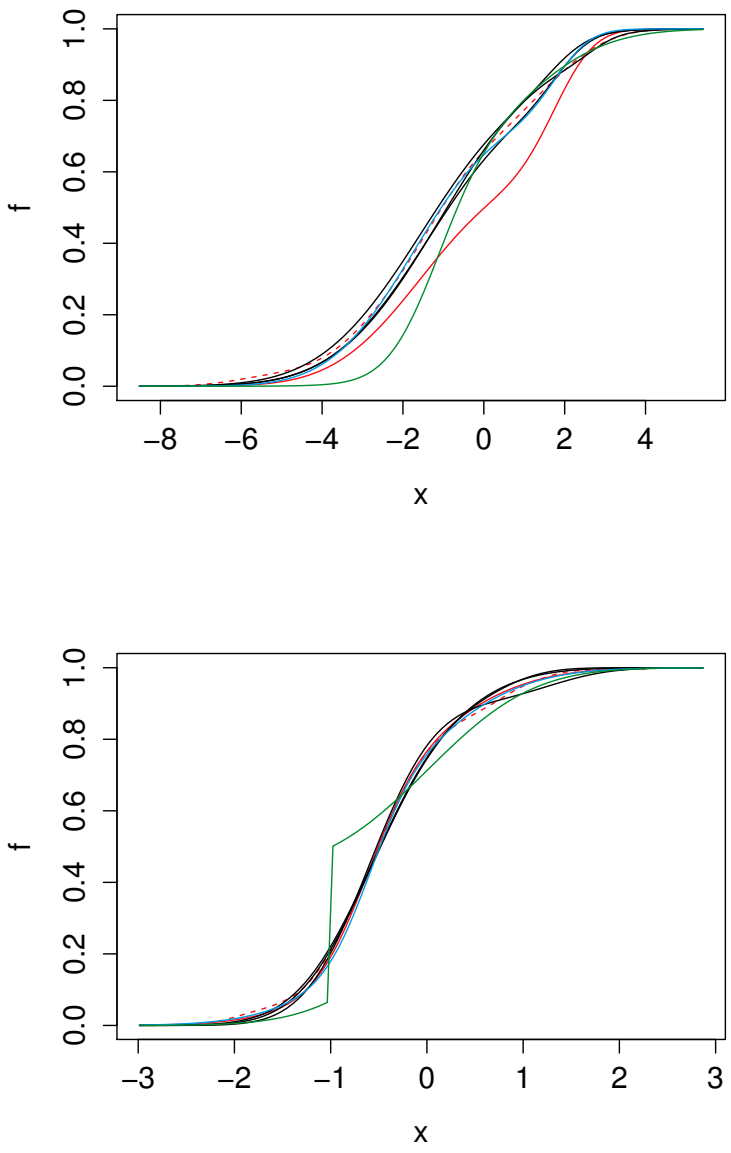


Figure 4.13: Top: Plot of $F(x)$, sample size: 50. Bottom: Plot of $F(x)$, sample size: 500 Both: True parameters in red, the EM estimate in blue, the MCMC estimate in green, the 3 best DLU estimates in black.

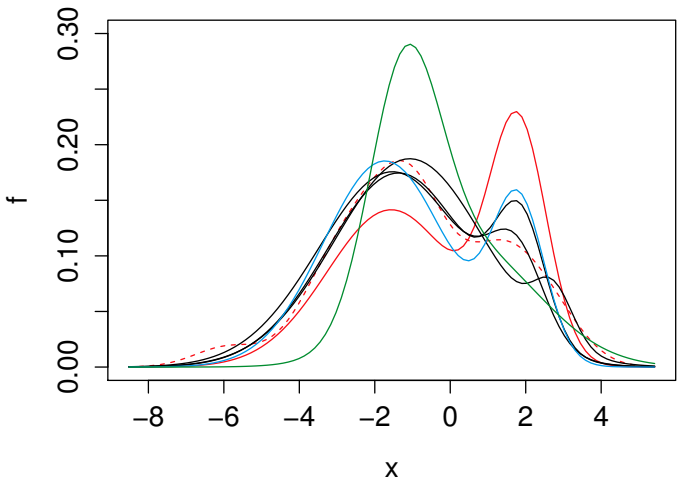


Figure 4.14: Plot of $f(x)$. True parameters in red, the EM estimate in blue, the MCMC estimate in green, the 3 best DLU estimates in black. Sample size: 50

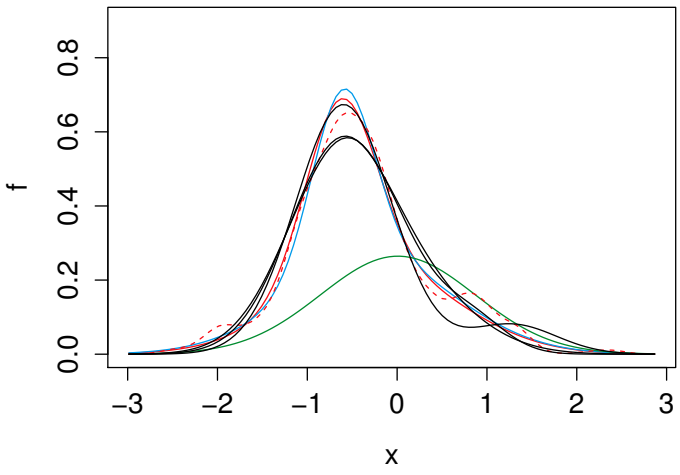


Figure 4.15: Plot of $f(x)$. True parameters in red, the EM estimate in blue, the MCMC estimate in green, the 3 best DLU estimates in black. Sample size: 500

Table 4.11: Example observations, sample size: 500. Top: True parameters of simulation. Bottom: Comparison of parameter estimates and the differences in log likelihood values.

	μ_1	μ_2	σ_1	σ_2	ε	$1-\varepsilon$
True values	−0.6308	−0.2483	0.3930	0.9211	0.4713	0.5287
Samples	−0.6128	−0.3637	2.4058	2.3010	0.4580	0.5420

	True	EM	MCMC	DLU1	DLU2	DLU3
μ_1	−0.6308	−0.6016	−0.9999	0.8437	−0.0863	1.2747
μ_2	−0.2483	−0.2766	0.0085	−0.5512	−0.6987	−0.5966
σ_1	0.3930	0.3442	0.0031	0.3723	0.7465	0.4946
σ_2	0.9211	0.9285	0.8606	0.6320	0.5480	0.5327
ε	0.4713	0.4118	0.4294	0.0745	0.4353	0.1000
loglik	−552.99	−551.89	−915.37	−576.15	−572.84	−571.38
diff	0.00	1.10	−362.38	−23.16	−19.85	−18.39

Table 4.12: Summary statistics for the differences in test log likelihood values for EM, MCMC and DLU

Difference	EM	MCMC	DLU1	DLU2	DLU3
Maximum	4.13	4.04	4.78	4.42	6.93
Minimum	−222.74	−10992.16	−1526.05	−1445.67	−9086.71
Average	−9.05	−112.50	−36.05	−41.62	−55.29
Variance	194.72	248062.08	14872.11	18825.34	130834.47

DLU prediction with the lowest prediction error and distance also has the highest log likelihood value.

These plots and tables could have been displayed for any of the 800 training observations, and are all based on data simulated as B.7 and computed as in the code B.5.

4.3.4 Time

System times were found in R, as described in section 3.8.2.

The elapsed times for 40 different lines are shown in the table 4.13. Summary statistics for these are found in the table 4.15. From the times and means it is pretty clear that there are differences in the how long time the methods use to return an estimate. It also looks like the runtime of EM and MCMC depends on the sample size.

An analysis of variance was performed for the differences between the methods to reveal clear indication that there are differences between the system times of the methods. Performing contrasts on the analysis, to find whether MCMC is different than the rest, EM different from DLU and fraction 24 and 25 are different from each other yield confirm that these four groups are different. The probability of observing these groups if MCMC has the same run time as the others is 4.9893×10^{-15}, for EM

Table 4.13: Table containing examples of system run time

Samplesize	EM	MCMC	DLU 2^{40-24}	DLU 2^{40-25}
50	0.10	3.12	0.73	0.39
50	0.00	2.78	0.75	0.37
50	0.07	2.85	0.70	0.37
50	0.04	2.87	0.71	0.37
50	0.02	2.76	0.74	0.42
50	0.02	2.83	0.73	0.37
50	0.01	2.76	0.70	0.37
50	0.02	2.59	0.72	0.36
50	0.05	2.91	0.72	0.36
50	0.00	2.67	0.73	0.36
100	0.02	3.31	0.72	0.36
100	0.00	3.29	0.75	0.36
100	0.03	3.05	0.73	0.37
100	0.03	3.20	0.72	0.37
100	0.03	3.28	0.73	0.37
100	0.01	3.46	0.71	0.36
100	0.02	3.17	0.74	0.37
100	0.01	3.18	0.70	0.37
100	0.01	3.40	0.71	0.36
100	0.03	3.21	0.73	0.36

Table 4.14: Table containing examples of system run time, continued.

Samplesize	EM	MCMC	DLU 2^{40-24}	DLU 2^{40-25}
200	0.25	3.99	0.73	0.38
200	0.05	3.85	0.73	0.36
200	0.00	4.16	0.71	0.38
200	0.27	4.04	0.75	0.38
200	0.14	3.98	0.73	0.36
200	0.09	3.88	0.71	0.36
200	0.25	4.04	0.72	0.37
200	0.02	4.09	0.70	0.37
200	0.10	4.07	0.70	0.37
200	0.04	4.08	0.70	0.40
500	0.42	7.19	0.73	0.37
500	0.06	7.18	0.73	0.38
500	0.00	7.11	0.71	0.36
500	0.19	6.99	0.75	0.38
500	0.42	6.98	0.71	0.38
500	0.20	6.96	0.71	0.36
500	0.42	6.79	0.70	0.38
500	0.19	6.77	0.72	0.38
500	0.02	7.19	0.71	0.37
500	0.00	7.07	0.71	0.37

Table 4.15: Means and standard deviations for the system times

	EM	MCMC	DLU 2^{40-24}	DLU 2^{40-25}
Mean	0.0913	4.2775	0.7208	0.3713
Standard deviation	0.1217	1.6682	0.0154	0.0122

Table 4.16: Means of system time per sample size

Samplesize	EM	MCMC	DLU 2^{40-24}	DLU 2^{40-25}
50	0.033	2.814	0.723	0.374
100	0.019	3.255	0.724	0.365
200	0.121	4.018	0.718	0.373
500	0.192	7.023	0.718	0.373

compared to the DLU methods 1.062541×10^{-31} and for difference between fraction 24 and 25 designs 1.7713×10^{-42}. These hypothesises of difference in the groups are confirmed from the table of average runtime values. Looking at differences between the sample sizes however, there are differences between the runtimes of different sample sizes in EM and MCMC. The analysis of variance of the system times in the different methods, the significance level for testing whether sample size has effect for system time of EM is 0.0017, same test for MCMC has a probability of 2.20×10^{-16}, while for the fraction 24 DLU design it is 0.7458. These probabilities tell the probability of these results if there are no differences in the system times. Looking at the average times for EM and MCMC, it is safe to assume that the system times increase with the sample size. The DLU look-up times are dependent on the fractions of the designs, how large the curve library is, but after that look-up time is constant. There is very little variation to the DLU method compared to the other two.

All system times reported in the comparison is run on the same system, under the same conditions, which is to say that all are run in one memory session, all the methods at the same time, one set of observations at the time. This means that more precise times and the time of child processes, which would have been interesting for estimation time of MCMC is not available. The run times are dependent on system, and show different results for each one.

Chapter 5

Discussion

5.1 Method

In order to evaluate the DLU method, it must be discussed and analysed. The work done with this implementation can be seen as two parts. There is the generation of the curve library that depends on the choice of the probability function, generator set and confounding patterns, and the number of levels in the parameters. Then there is the estimating of parameter sets using the direct look-up method, and comparing it to established methods. To be able to evaluate designs and generator sets the error of look-up is used over a large number of observations for various designs. To compare the DLU to the EM and MCMC methods, the design that achieved overall best results for look-up error was used.

5.1.1 Parameters and MBR design

The definitions of the parameters define the entire method. The choices made regarding the number of levels of the parameters and limits are therefore important. The accuracy of the predictions depend on the number of levels in the parameters. Increasing the number of levels while keeping the parameter value limits increases the density of the parameter values. To illustrate; the parameter values of the mixing parameter ε are seen as $\{0.00, 0.00196, 0.00392, 0.00588, 0.00784, \ldots 0.50\}$, hence any changes that are smaller than these in the true parameters are beyond the reach of the DLU method. There is simply not enough data to represent it. This is true for all the parameters. The number of levels for the parameters is chosen to achieve a reasonable level of accuracy. Smaller differences than these are difficult to find based on random samples of a probability distribution. The number of levels in each parameter is described in the table 3.1 and their

values are described in the section 3.2. The lengths are chosen based on value density and the area of parameter values to represent.

The MBR design method is implemented to transform the multilevel design factors into binary factors. Due to the restriction in the number of levels in a parameter, changing the length by just one step will either double or halve the initial number of levels. The number of parameter levels grow quickly when increasing the density.

The algorithm as described in Martens et al. [2010] is changed in scope from the original implementation. There are several changes that have been made during the development of this method, and among them are: adding support for specifying the order of the bit factor in the confounding patterns, specifying the generators themselves, and removing all restrictions regarding size of the design. This was necessary to provide large enough designs. This code is available in appendix B.1.

Designs of any size can be created, which is particularly important with the restriction of the parameters. The parameter restrictions have effect in the estimation problem of the mixture models, for instance the restriction of σ_i removes three quarters of the design. Specifying the order of the bit factors in the design is used to avoid full confounding of one design factor with another design factor. As all the bit factors can be divided into groups based on which design factor they represent this might have an impact on the spatial span of the design factors in the fractional design. A possible way to manipulate the confounding patterns is to control what bit factors confound with other bit factors. The options for combining different bit factors for a confounding pattern are many.

With the changes in the creation of the MBR design, the generators have to be specified, as there is no option to use known designs. In order to create a design, decisions about the generators must be made. This is a necessary change in order to create large designs, and the updated method is made as an alternative method to support the creation of larger designs.

Increasing the number of levels in the parameters may increase the accuracy of the predictions. In Isaeva et al. [2011b], it was proven effective to increase the density, and it worked well to identify the correct function. This also provided a more precise estimation of the function parameters. However, increasing the lengths of the parameters to achieve more precision is not always a good idea. The number of unique combinations of different parameter values will increase, and it is needed to either use a higher fraction confounding pattern or to increase the number of curves allowed in

the library. Increasing the fractions, creating more confounding variables can lead to issues with confounding patterns.

From experiments with a denser parameter space, it seems that the parameter estimates are not better for the library with larger number of levels per parameter. The average prediction error for the denser library is 0.00024. For comparison the means of the other libraries are presented in 4.2. The average prediction error is just as for the other libraries of random generator sets. From the table containing the average log likelihood differences, 4.6, the library with denser parameters than the others does not stand out compared to the other averages, and it does not differ statistically. Increasing the density without increasing the number of curves in the database, by increasing the fraction and number of generators, does not reduce the average difference in log likelihood values, and it is not significantly different from the differences in the other curve libraries of the same size. (See table 4.5) This indicates that increasing the number of levels in the parameters without increasing the size of the library has little effect. Trying to increase the parameter space density to achieve better results may give smaller errors and more confounding.

Log likelihood evaluation of the estimated parameters show no effect of fraction in the design or the levels of the parameters. This is for the average of a large number of observations. For very specific data input problems there might be a larger dependence on the parameter values. All observations here come from a larger set of x values than 0 to 1, and processing on the x-axis is performed. This magnifies the discrete jumps in the parameter values if the processing parameters are large.

The values of σ_i^2 are dependent on the corresponding value of μ_i. There are two strategies to deal with this; either perform line generation for the lines that are valid combinations of μ_i and σ_i^2, or perform preprocessing on the library curves to make the library curves fulfill the necessary conditions for a distribution or density function.

Avoiding preprocessing in the database reduces the number of library curves to a quarter of the combinations in the design. This make tools for evaluating design more or less useless, as there is no way to know if the alias structure remains the same when three quarters of the design is removed. When the parameters have the number of levels as defined in table 3.1, it translates to a large number of binary factors. The parameters are of length 8, 9, 9, 7 and 7. In total, 40 binary factors are needed. Still, the number of runs in the library should not be much larger than 2^{14}, to avoid look-up taking to long to run. The size of the generated library will always be a

power of two, such as $2^{13} = 8192$, $2^{14} = 16384$, $2^{15} = 32768$, etc. If a design of about 2^{14} is the desired result, a $2^{16} = 65536$ design has to be generated.

Preprocessing is done on the databases described by Isaeva et al. [2011b]. In the current implementation of DLU for mixture models the parameter values are chosen to fulfill the necessary conditions at creation. If preprocessing is used, generating a design would be faster, as the original design size does not need to be four times the size of the desired library size. It does however create a larger need of processing the curves, including the library, the inputs, and the desired outputs. Curves can be generated in the correct range and have been chosen in this implementation. A future comparison of the two strategies for different behaviour could be of interest, to study potential differences between the strategies of library curve selection.

5.1.2 Generators

Studying the aliasing patterns of the generators in the general design was not going to work for this implementation of the DLU. The number of combinations still to be included in the library change somewhat with every generator set, and this is why the approximate size of the design is used. The spatial limits (figure 3.3) on the parameter combinations remain the same, but the placement of the included combinations differ. The spatial resolution of the design will differ in each library, and the number of points included in the valid parameter space will be a bit different every time.

Traditional rules for finding good designs are not necessarily valid for MBR design, as the interesting factor is the prediction error of the estimation. Estimation is done on the design factors, while most rules for finding designs will in the MBR design only optimize the bit factors. Confounding patterns of the bit factors are of little interest compared to the prediction error of the final look-up values, which are based on the design factors.

The averaged mean square errors for look-up of the new observations for different confounding patterns has been investigated to choose a design. The errors for designs of the same size are calculated, and can be used to decide between designs. The results of prediction error comparisons show that the error is higher in general for the odd and full confounding patterns than for the confounding patterns with smaller generators, as seen in 4.2. The variation between different sets of generators in full confounding patterns is also larger than for other groups, although this might be

random chance. The generators are more evenly distributed amongst the bit factors. It seems however that the few randomly chosen generators in the odd and full confounding patterns have more influence over the final library than when all generators vary. The analysis of variance (table 4.3) and contrasts reported in section 4.2.1 does provide reasons for choosing a library from a design of random generators. Both odd and full confounding patterns have significantly higher errors than the random generators. These patterns might have better use in smaller designs, where the dimensions are more traditional for fractional factorial designs than they are here.

As stated in Thalberg [2011] where the optimization of MBR designs were tested extensively, MBR designs rely upon the choice of generators. The different generator sets might optimize well for different problems, and the confounding patterns should be chosen based on the estimation problem. Testing and optimizing was performed to find designs that worked well for different estimation problems, all for relatively small designs. This leads to conclude that the traditional confounding patterns do not work well from bit factors back to ordinary factors, and that by doing this one changes the traditional rules of fractional factorial designs.

Differences in log likelihood values of the estimated parameters are used as a more reliable measure of the generalized prediction error. From the table 4.4 there appears to be slightly different generators that are optimal. There are no significant differences in the log likelihood values when including all the tried generator sets. From the average values seen in table 4.6, the lower fraction seem to have better results than the higher fractions, but the differences in means are too small compared to the variance of the deviances to conclude any certain differences.

When there is no significantly better result to choose, and one from other sources know that the confounding patterns will work better or worse depending on the estimation problem, how do you choose? To validate the DLU method against the EM and MCMC methods, a generator set that does well for both prediction error and average log likelihood difference is chosen.

Examples of results and comparisons were performed on one of these libraries, the one that gave the smallest error of prediction, the second largest log likelihood deviance, and from a group with less variation than the others. For this implementation of DLU it was tried to find a design that will in average yield good results over many different estimation problems. Designs are chosen based on average values and difference in groups, rather than which designs were best for one random estimation

problem. The design with the least error was chosen to use for comparisons, even though the groups did not differ significantly from each other. The random design with the lowest error of prediction was chosen according to optimality criteria. As MBR designs should be chosen depending on the estimation problem and this was the random design that works best for this estimation problem of the randomly generated designs that have been tried here, it was chosen for the method comparisons.

Investigating the alias patterns of the bit factors and how they impact on the original factors has not been properly tested, neither earlier nor here. With the parameter lengths chosen in this implementation, the possibilities of aliasing factors and confounding patterns are so many and so huge, this should be tested systematically in a controlled setting, where any possible effects can be studied. Finding possible effects of the aliasing bit factors may lead to general rules for choosing confounding patterns and generators for MBR designs.

5.2 Compression

5.2.1 f or F

Looking at the results from the investigation of the differences between compression results for $f(x)$ or $F(x)$, presented in section 4.1.1, the difference is apparent and stable. The scree plots, figure 4.7 and 4.9, figure 4.8 and 4.10, and the table containing the number of principal components needed to explain 99.9% of the variability (4.1), show the difference in proportion explained by each component.

The data in table 4.1 shows a summarized version of compression results. Each row in the table represent a different setting of the generator choices, different fractions for the factorial design and different bit sizes of the generators.

If one were to perform an analysis of variance on these data there would be a definite difference between the performance of the density function $f(x)$ and the distribution function $F(x)$, in compressing the database. There is clearly large differences between the groups compared to the variation inside the groups, and for both centred and uncentred analysis, the density function has drastically weaker compression results than the distribution function.

To continue using the density function is no further help than to use the complete curve library, since almost no amount of data can be reduced using

the principal component analysis. The compression is performed to allow storing the curve library using only a small set of the principal components, and when this is not achievable through using the density function, it is decided to use the distribution function $F(x)$ for further implementation.

5.2.2 Centring of the data

Principal Component Analysis can be performed on centred data or the original data. This affects the computation, the scores and the loadings, and it is needed to retain the center of the library for use in performing look-up of new observations. This does not change the look-up results for the same set of observations.

The number of principal components needed to explain 99.9% of the variation in the database appears to be a lot less for uncentred data than for centred data. It looks like a great deal of the variation in the data can be explained by the mean of the columns of X. For uncentred data it is not necessary to include enough principal components to have 99.9% of the variability covered. If 99.9% of the variability in the data can be explained using the k first components of the uncentred matrix, that will also be valid for the centred data. Since so much of the variation is removed when subtracting the mean, the remaining proportions grow larger. Compression can be performed and retaining only enough of the principal components to explain 99.9% of the variability in the uncentred data to reduce the number of components necessary. It seems that a lot of the variability is removed when centering the variables before compression, and that the number of principal components to include using centred columns of data can be discussed. It is believed that when only k components are needed to explain 99.9% of the variability in uncentred data, k can be used to explain 99.9% of the variability of the original data, even though analysis is run on centred data. This is because the average would be very apparent in the first component principal component, when subtracting the mean, most of the first principal component is removed as well and there is less variability in the data, but a bit more for each remaining component.

5.2.3 Scoreplots and loadingplots

The scoreplots as shown in 4.1 through 4.4 are very different from the scoreplots of other data. As this is design data in a constricted parameter space, the scores have very distinctive limits and unusual shape. From the scoreplots it is apparent that the shape of the scoreplots differ in DoE data from that of non design data. There are clearly defined shapes. Between

centred and uncentred data analysis it is seen that the direction of the principal components differ. The amount of variance explained for each component is also stated in the plot, and it is apparent that the first component is responsible for more of the variability in the uncentred data than it is for the centred data.

Comparing the scoreplots for data from the distribution function $F(x)$ to that of the density function $f(x)$, it is seen that the density function explains significantly less of the variability per component, and that the directions are less defined.

The centred data of the distribution function $F(x)$ has loadingplots, see figure 4.5, where the first component (shown in black) show signs of being half a period of a periodic function. The uncentred data is a whole period, or twice as much information as in the first period. The data for the density function $f(x)$ in figure 4.6 are less defined in their shape, but the periodic trends are there. In these plots as well, the centred data to the left has double the information of the uncentred data to the right.

The loadingplots show connections to the Fourier transformation. This connection is for instance stated by Wall et al. [2003]. As stated in 2.26 and 2.27 it is seen that the Fourier expansion of a matrix can be split into parts similar to the singular value decomposition. The loadings seem equivalent to the Fourier expansions from math and plots, and the connection is stronger for components explaining a large proportion of the variance, where the plot of the loadings are more continuous curves. The connection to the Fourier is more apparent for the centred data than for the uncentred data, the left versus the right of the figure 4.5.

5.3 Performance results

5.3.1 Direct Look-Up

Results for evaluating the parameter estimations show that there is not necessarily a link between the least distance and the best log likelihood estimate for independent test data. This raises questions about the selection criteria in the curve library. A different measure than Euclidean distance in the score space might provide a better selection criteria. The speed of look-up is directly tied to the selection criteria, computing the criteria and finding the best one. To improve this a better programming search structure could be implemented.

Parameter estimation of the DLU could clearly be better. The overlap

between an equal mixture and a univariate model is sometimes very apparent. The estimations are very dependent on the actual values present in the sample, if the random sample differ from the true distribution.

5.3.2 Results and reported lines

Examples of the estimated parameter sets are seen in tables 4.8 and 4.9, and the corresponding plots 4.12,4.14 and 4.15. It seems that estimated parameter sets that visually fit very well to the data can have very different values. The mixture model depends on several parameters, and many of these will compensate for each other. Many of the mixtures will be estimated as a mixture with a very low weighting parameter ε or variances σ_i^2. Implementing a minimum value for the mixing parameter and the sigma to include in the mixtures will possible reduce this problem and create a better library of curves.

It is clear from the log likelihood values presented in table 4.10 and table 4.11 that the method is not optimized to find the largest log likelihood values when selecting estimation solutions. It is often that the second or third set of estimated parameters has a better log likelihood value than the first. The likelihood values decrease as the sample size increases, and the difference from DLU to the others is greater the larger the sample gets. It is observed that estimation methods can give higher log likelihood values than the true parameters does. The best look-up set of parameter estimates do not necessarily give the best log likelihood values. The log likelihood values are not used as a selection criteria for the estimated parameter sets. The curve library is implemented using Euclidean distance in the score space as selection criteria, which can be classified as a nearest neighbour estimator. The inconsistency between results in distance and log likelihood evaluation of the estimation indicate that the score distance is maybe not the best tool for selecting model predictions. Investigating alternative selection criteria for what qualifies as the best model can be a good idea, to ensure a better connection between good estimated parameter sets and the best curves in the library.

5.3.3 Comparison

Performance results in prediction error and log likelihood values are recorded for both DLU and EM and MCMC over the same set of lines. The results are summarized for two examples in table 4.10 and table 4.11. A

visual comparison of these two example results are found in the figures 4.13, 4.14, and 4.15.

For the first of these two examples, this is a mixture distribution with two defined components, different means and standard deviations. This is an example of a small sample situation, and the sample is not necessarily very well fit to describe the model. The offset and slope parameters for the x-axis are recorded as a means to shift the estimated parameters to the x range of the observations. Looking at the parameter estimations of EM, MCMC and the three first estimated parameter sets from DLU, it is clear that despite the least distance the first DLU result is not the best. Of the three DLU results it performs the worst in this case. The difference in log likelihood value range the methods from EM, to DLU and lastly MCMC. There are great variations between the DLU estimates, especially for the mixing parameter ε. This is of some concern, that the DLU seems unable to differ between mixture distributions and ordinary normal distribution.

These trends are also present for the second example, of table 4.11 and plot 4.15. Visually, it looks like a better fit than the first example with a smaller sample , though not in the difference of log likelihood values. Again it is seen that the proposed DLU solutions vary a lot, and especially for the mixing parameter ε. The estimated parameter values depend on each other, if the mixing parameter is very small, one of the distribution components will have to explain almost all of the changes, and this may affect the estimate of expectation and standard deviation for this component. We see that the sample has great influence over the function parameters. The fit of the library curves to the non parametric distribution estimate is important in the estimation of the parameters. The non parametric distribution is more accurate with larger sample sizes, and so the accuracy of the parameter estimates should increase with sample size.

The two examples presented in the appendix, tables A.3 and A.4, figures A.2 and A.3, show that for mixtures where the mixing parameter is so low that it is nearly not a mixture at all. The estimated parameters vary greatly, between mixture distributions and ordinary normal distributions of one component.

The DLU method is being compared to methods specifically created to solve the mixture model problem fast and efficiently. One strength of the first implemented DLU was that no previous knowledge of what the problem represented was needed. It could do parameter estimation for different types of mathematical functions faster than precise estimation of parameter values for specific methods, and give an indication of what

function type the data represents. In this case the function type is known, and the method is implemented for only a mixture of two normal distributions. The precise parameter estimation is not the strength of the DLU, causing it to lose a lot of the original advantages in this problem. If the distribution database had been extended to include not just two component mixtures, but three or four components, ordinary distribution functions or combinations of different distribution types, the DLU might have worked as a better way to differ between these types of distributions.

For precision of predicting the parameter values however, a very dense parameter space with many lines is needed, and that takes significantly more time than running a specialised method such as the implementation of EM used here, which is optimized for estimation of this exact problem. To increase the accuracy of the parameter estimations the number of parameter levels of for instance μ_1 and μ_2 can be increased. Increasing the number of levels allow smaller changes to be found. Together with increasing the number of levels, also increasing the size of the curve library is necessary to achieve better accuracy. Choosing a lower limit for the standard deviations σ_i in the curve library avoids the curves with very steep peaks in their density. These curves tend to estimate the sample with one very weighted part, and one distribution with almost no variation and very little weight. To implement a lower bound for either the σ_i's or the ε can help avoid these estimations. The figure 4.11 shows that a large number of randomly picked curves from the databases have small values for σ_i, they have sudden changes in direction, which indicate steep points in the mixture. Computing the library as it appears, with preprocessing of the curves rather than removing the parameter combinations that do not fulfill the conditions, might reduce part of this. The overall standard deviations would be larger, and the curves would have less distinct turns.

The curve libraries are implemented using the distribution function $F(x)$. Looking at example distribution curves from a curve library of distribution curves as in 4.11 some of the issues with the parameters are represented. The lines are at the densest in the center of the plot in the area of $F(x) = x$. The lines seen in the upper and lower edges of the plot often have sharp turns and points in the values. These are examples where the mixture parameter is either very low or one or both of the σ_i's are very close to 0. The same will be apparent for the density functions, and will be shown as a very distinctive peak in the density values. These peaks are not really representative of the mixtures that we want to estimate using methods like these. A stricter control of the values for σ_i and ε to include in the curve library would be desirable to solve some of these problems. A limit for ε could be decided depending on the sample size. It could be

decided that the mixing parameter to look for is at least 10-20% to avoid the estimated models where one mixture component explains almost all of the data.

The distribution function $F(x)$ was chosen for the curve library, as it allows compression so much better than the density function $f(x)$ does. From the number of components needed to retain 99.9% of the variability, there is almost nothing to gain by performing a PCA on the library of $f(x)$ in order to reduce the size of the library. However, using the density function $f(x)$ might have some advantages over the distribution function $F(x)$. The shape of the density function differs more based on the function parameters than the distribution function does. The density function can help avoid estimation results where one standard deviation is very small, or the mixing parameter is close to 0 or 1. The MSE of a density curve estimate of a curve with a very specific peak compared to the rest will be much higher than the same measurement for a distribution curve. When estimating mixtures it is normally assumed that both mixtures will be present in more than just a single point.

The performance is also dependent on the simulation parameters. The log likelihood are worse if the variation in the dataset is larger, and the DLU is better at estimating the small samples simulations than EM and MCMC. Part if this is probably due to the fact that DLU takes any input data and estimates a 100 points density curve, to compare in the database. The density estimate have to be of a standard length, same as in the curve library, but there is no reason why this has to be of exactly 100 points. EM and MCMC works directly on the observations that is given as input, and gets more and more precise the larger the sample size is, while DLU still performs look-up on the 100 density points that are estimated at the beginning of the look-up process. A more efficient density estimate, of not necessarily 100 points in length could give a better approximation to data.

5.3.4 Time

The DLU library of this implementation is relatively large. The larger the library, the longer time it takes to search through it to find the closest distribution curves.

The implemented EM algorithm is a very fast algorithm. It is optimized to give very fast and precise results for estimation of mixture model of two normal distributions. MCMC for R using OpenBUGS is dependent in the computer system it is run on. It calls external software OpenBUGS and performs the analysis, and time elapsed can therefore get very large.

As seen in the table 4.13 an example run of a look-up estimation of a

processed observations curve at a library size of about 2^{14} curves use less than a second. Many of the EM estimations will appear as to have taken no time at all when results are found on a Windows system due to lack of precision. If the fraction is reduced by 1 and the library size is reduced to half of this, the look-up time of the same curve is also halved. The look-up times of DLU are relatively constant and depends on the size of the MBR design. The table 4.15 contains the means and standard deviations of the look-up times over these 40 observations. It is clear that DLU is very consistent in the look-up times. MCMC has the highest means and deviations. In the table 4.15 the means are found for the different sample sizes. This confirms the idea that MCMC and EM are dependent on the sample sizes while performing estimation, and that estimation times get larger when the sample size increases.

The first strategy of reducing look-up time for when the curve library gets large is to reduce the number of operations performed. Testing to see how small the curve library can be made before it affect the parameter estimation too badly is also an option. A different selection criteria to find the distance between the observations and the library might be easier to sort or find the best best set of observations. To find the single best line there are programming strategies to reduce search times, but if they are efficient when finding the i.e. 10 best, is unknown. Many search algorithms are very fast, but often assume a logical way to sort the data that is not necessarily present with the library curves.

5.3.5 Weaknesses

The parameter estimations differ greatly from one line to the next, even though both visually and in score distance they are very similar. This shows one of the problems with mixture distributions, that unless there are clear peaks in the density, the shift between the distributions and the parameter values of the individual distributions can compensate for each other. The estimations are very dependent on the observation sample. The look-up method finds a distribution that is very close to the sample data. If the random sample for estimation is a poor representation of the true parameters, the estimated parameter results are not going to score well in comparison using the log likelihood values for a independent test set.

This implementation does not take advantage of the fact that this method is extremely generalizable. There is no need of priors, starting values, iteration limits or knowledge of probability distributions. If the

database was expanded properly with different model options, more combinations and distributions, the DLU method would be able to take any input observation data and provide a set of suggestions to which distribution might explain these data. Based on these results the plausible distribution model could be chosen. E.g. if there is a suspicion about data following a normal distribution and a predicted curve suggest a Poisson distribution, the user would check again or look for the fit of normal distributions.

5.4 Uses of DLU

There are many problems on which DLU can be used with success. The diversity of this model allows the library to include any kind of function that is desired to perform the look-up method for.

As an example, a mixture of binomial distribution is presented in Snipen et al. [2009]. This experiment estimates the unknown distribution parameters of the mixture binomial distribution to decide the number of unknown genes. The mixture could be estimated using DLU with a library including mixtures of binomial distributions to find the parametric estimate of the distribution.

The DLU method provides a very generalizable method. The curve library can be extended or applied to any form of mixture model. There could be mixtures of many distributions or of different distributions, all included in the same collection of libraries or metamodel. The look-up method has the ability to identify and differ between different mathematical functions and model structures, and this is one of the reasons why it is a very effective model for the non linear model situation.

Chapter 6

Conclusion

In this thesis, a working DLU method for parameter estimation of a mixture of two normal distributions is implemented. There is now an non-iterative method for parameter estimation. It does not perform as well as we would like compared to established methods such as EM, but do provide better estimates than the MCMC algorithm does.

However this implementation feels like a beginning. It feels like the beginning of a larger collection of libraries or a metamodel for many different mixtures. The current results show that the idea of using the DLU method to estimate mixture models work.

The size of the current problem make it difficult to find the best way to optimize it. For optimizing the generators of the design there are so many different options and possibilities. There are so many variables that could be optimized, and the initial results prove that there is potential in the method. A curve library based on a general design can be generated, compressed and stored, ready for look-up. Compression and storage have been optimized based on results. Look-up is implemented, with room for improvement, but it predicts solution curves of input observations. The look-up results are compared with established methods over multiple observations to evaluate the performance of the DLU method. This has been an experiment in developing methods, analysing results and changing the method based on experimental results.

A future implementation would do well with some changes. As initial suggestions; increasing the size of the curve library, together with implementing a faster search algorithm to avoid estimation time being too long. If limits were included on the values of σ_i and ε it could ensure proper mixtures, not just a single distributions with an extra peak. To find a way to perform the library compression of the density function $f(x)$ instead of the distribution function $F(x)$ might also allow detection of these problem

curves, removing them from the curve library prior to performing look-up. Extending the library to a larger collection of libraries, including distributions of only one component, mixtures with parameter limits, combinations of different distributions, etc. might provide a look-up library that can test for different models and combinations in one estimation. This can propose the best alternatives, while traditional methods would have to be implemented for every possibility that is of interest. Much of the comparison are made on the log likelihood values of the predictions, which is not the selection criteria for the curve library. An idea would be to try this using a maximum likelihood estimator for the curve selection instead of the score distance. A different selection criteria may require a different, faster search structure, allowing faster look-up results.

Performing work in explorative areas opens more doors than it closes, it poses more questions than it answers. Taking this further, it is really expanding the curve library into a metamodel of libraries that is of more interest than the single curve library. This provide uses that might work better for the DLU method than precision parameter estimation of a mixture of two normal distributions.

Bibliography

Tatiana Benaglia, Didier Chauveau, David R. Hunter, and Derek Young. mixtools: An R package for analyzing finite mixture models. *Journal of Statistical Software*, 32(6):1–29, 2009. URL `http://www.jstatsoft.org/v32/i06/`.

Peter J. Bickel and Kjell A. Doksum. *Mathematical Statistics basic ideas and selected topics vol.1.* Pearson Education, Inc, 2007,1977.

José G. Dias and Michel Wedel. An empirical comparison of em, sem and mcmc performance for problematic gaussian mixture likelihoods. *Statistics and Computing*, 14:323–332, 2004.

Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning.* Springer, 2009.

Julia Isaeva, Solve Sæbø, John A. Wyller, Sarin Nhek, and Harald Martens. Fast and comprehensive fitting of complex mathematical models to massive amounts of empirical data. *Chemometr. Intell. Lab. Syst.*, 2011a.

Julia Isaeva, Solve Sæbø, John A. Wyller, Olaf Wolkenhauer, and Harald Martens. Nonlinear modelling of curvature by bi-linear metamodelling. *Chemometr. Intell. Lab. Syst.*, 2011b.

Richard A. Johnson and Dean W. Wichern. *Applied Multivariate Statistical Analysis.* Pearson Education Inc, 2007.

David C. Lay. *Linear Algebra and Its Applications.* Pearson Education Inc, 2006.

Hedibert Freitas Lopes. Univariate mixture of normals: Mcmc and em algorithms. *Applied Econometrics, Spring 2005*, Spring 2005. Graduate School of Business University of Chicago.

Harald Martens, Ingrid Måge, Kristin Tøndel, Julia Isaeva, Martin Høy, and Solve Sæbø. Multi-level binary replacement (mbr) design for computer experiments in high-dimensional nonlinear systems. *Journal of chemometrics*, 2010.

Douglas C. Montgomery. *Design and Analysis of Experiments*. John Wiley and Sons, Inc., 2009.

R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2011. URL `http://www.R-project.org/`. ISBN 3-900051-07-0.

Lars Snipen, Trygve Almøy, and David W Ussery. Microbial comparative pan-genomics using binomial mixture models. *BMC Genomics. 2009; 10: 385.*, 2009.

Sibylle Sturtz, Uwe Ligges, and Andrew Gelman. R2winbugs: A package for running winbugs from r. *Journal of Statistical Software*, 12(3):1–16, 2005. URL `http://www.jstatsoft.org`.

Hege Grøstad Thalberg. A comparison study of different optimizing criteria and confounding patterns for multi-level binary replacement and other designs used in computer experiments. Master's thesis, NTNU, 2011.

Kristin Tøndel, Arne B. Gjuvsland, Ingrid Måge, and Harald Martens. Screening design for computer experiments: metamodelling of a deterministic mathematical model of the mammalian circadion clock. *Journal fo Chemometrics*, 24:738–747, 2010.

Michael E. Wall, Andreas Rechtsteiner, and Luis M. Rocha. Singular value decomposition and principal component analysis. *A Practical Approach to Microarray Data Analysis. D.P. Berrar, W. Dubitzky, M. Granzow, eds. pp. 91-109, Kluwer: Norwell, MA (2003). LANL LA-UR-02-4001*, 2003.

Appendix A

Extended results

A.1 Prediction error

Table A.1: The full table of DLU look-up results including errors and distances for the top ten estimated parameter sets for a small sample size

	μ_1	μ_2	σ_1	σ_2	ε	Error	Distance
True	1.2852	−1.4165	2.7484	1.7850	0.0482		
1	−1.6256	−1.9842	1.2196	1.5416	0.2412	$1e-04$	0.0792
2	−0.8127	−1.9603	1.3162	1.4289	0.0157	$1e-04$	0.1027
3	−1.2191	−2.3906	1.1714	1.4451	0.4137	$1e-04$	0.1107
4	−1.3626	−2.0320	1.8635	1.3324	0.2157	$2e-04$	0.1226
5	−2.5819	−1.6973	1.4611	1.4289	0.2784	$2e-04$	0.1313
6	−0.7409	−2.0080	1.7026	1.3807	0.0569	$2e-04$	0.1328
7	4.1363	−1.8886	0.0606	1.4129	0.0059	$2e-04$	0.1355
8	−3.5621	−1.7690	1.1553	1.3002	0.0765	$2e-04$	0.1457
9	−1.3626	−1.8886	0.4309	1.4773	0.0431	$2e-04$	0.1463
10	−3.6339	−1.4582	1.0747	1.1392	0.2294	$2e-04$	0.1519

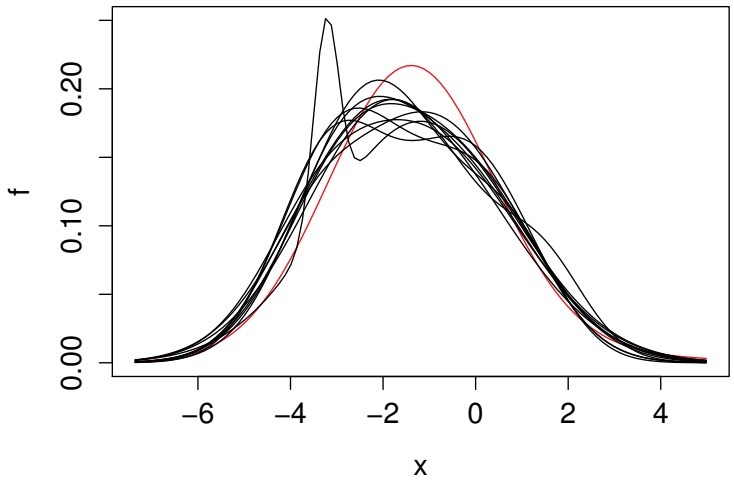


Figure A.1: Plot displaying the density function using the true parameters (red) and the ten predicted parameter sets of DLU (black). Sample size: 50.

Table A.2: The full table of DLU look-up results including errors and distances for the returned ten curves of a large sample size

	μ_1	μ_2	σ_1	σ_2	ε	Error	Distance
True	−0.1698	−2.9525	1.5556	2.8477	0.0322		
1	−2.3821	−3.1019	2.4481	3.0944	0.2412	$3e-05$	0.0562
2	−0.7505	−3.0539	2.6420	2.8682	0.0157	$4e-05$	0.0664
3	−1.5662	−3.9176	2.3512	2.9007	0.4137	$4e-05$	0.0664
4	−1.8542	−3.1978	3.7406	2.6745	0.2157	$5e-05$	0.0696
5	−4.3016	−2.5260	2.9329	2.8682	0.2784	$5e-05$	0.0736
6	−0.6063	−3.1497	3.4176	2.7714	0.0569	$6e-05$	0.0766
7	9.1836	−2.9100	0.1216	2.8360	0.0059	$6e-05$	0.0767
8	−6.2691	−2.6699	2.3190	2.6098	0.0765	$6e-05$	0.0773
9	−1.8542	−2.9100	0.8650	2.9653	0.0431	$7e-05$	0.0839
10	−6.4133	−2.0460	2.1573	2.2866	0.2294	$8e-05$	0.0915

A.2 Comparisons

Table A.3: Top: True parameters of simulation. Bottom: Comparison of parameter estimates and the differences in log likelihood values. Sample size: 50

	μ_1	μ_2	σ_1	σ_2	ε	$1-\varepsilon$
True value	1.2852	−1.4165	2.7484	1.7850	0.0482	0.9518
Sample	−0.6460	−1.7705	NA	2.4057	0.0200	0.9800

	True	EM	MCMC	DLU1	DLU2	DLU3
μ_1	1.2852	0.4287	−1.7598	−2.9644	−2.7492	−3.2274
μ_2	−1.4165	−2.3634	−1.0027	−0.7888	−0.2629	−1.1952
σ_1	2.7484	1.3648	3.6923	1.2036	1.4129	0.2699
σ_2	1.7850	1.2821	2.0877	1.8313	1.6060	2.0406
ε	0.0482	0.3492	0.4574	0.2961	0.4706	0.0980
loglik	−103.02	−104.37	−108.55	−103.32	−102.71	−102.59
diff	0.00	−1.3530	−5.5349	−0.3034	0.3021	0.4218

Table A.4: Top: True parameters of simulation. Bottom: Comparison of parameter estimates and the differences in log likelihood values. Sample size: 500

	μ_1	μ_2	σ_1	σ_2	ε	$1-\varepsilon$
True value	−2.9525	−0.1698	2.8476	1.5556	0.9678	0.0322
Sample	−2.9125	0.0555	2.3478	2.5750	0.9680	0.0320

	True	EM	MCMC	DLU1	DLU2	DLU3
μ_1	−0.1698	−3.5755	−2.0000	−2.3821	−0.7504	−1.5661
μ_2	−2.9525	−2.5619	−2.8769	−3.1019	−3.0539	−3.9176
σ_1	1.5556	3.6569	0.0059	2.4481	2.6420	2.3512
σ_2	2.8476	2.4088	3.0804	3.0944	2.8682	2.9007
ε	0.0322	0.3139	0.1188	0.2412	0.0157	0.4137
loglik	−1205.1	−1207.0	−1270.8	−1207.6	−1206.0	−1206.5
diff	0.00	−1.9676	−65.7174	−2.5528	−0.9617	−1.3898

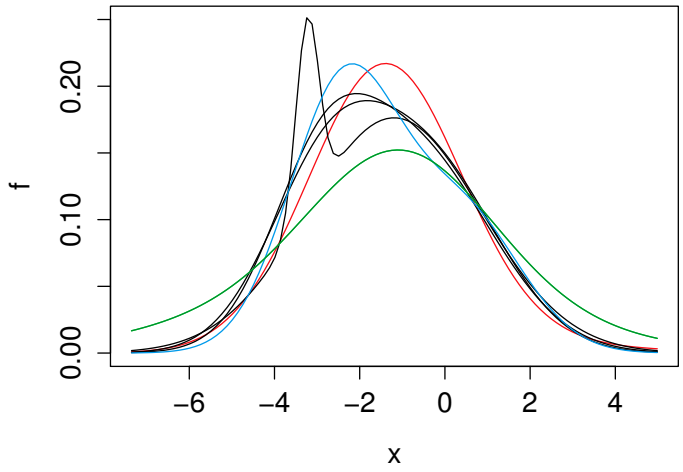


Figure A.2: Plot of $f(x)$. True parameters in red, the EM estimate in blue, the MCMC estimate in green, the 3 best DLU estimates in black. Sample size: 50

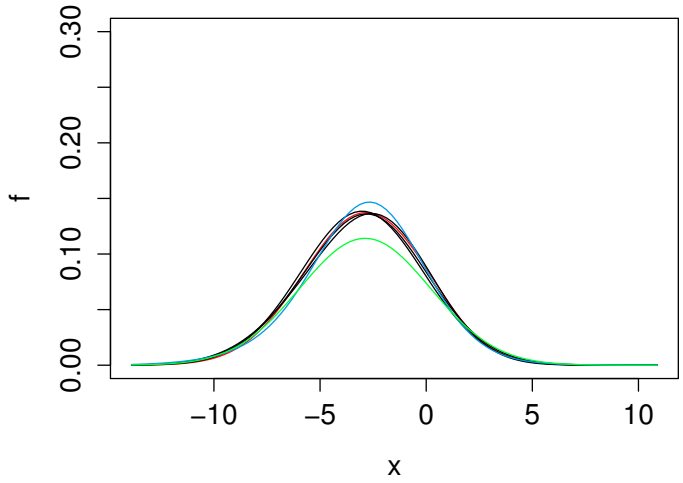


Figure A.3: Plot of $f(x)$. True parameters in red, the EM estimate in blue, the MCMC estimate in green, the 3 best DLU estimates in black. Sample size: 500

Appendix B

Simulations and code

B.1 Code for DLU

The code for generating the necessary information to perform DLU estimation.

Listing B.1: This code creates the multi level binary replacement design

```
#Multi-level Binary Replacement Design

#Arguments
#l2levels = A vector indicating the number of log2
#-levels for each factor. E.g. c(2,3) means 2 factors,
#the first with 2^2=4 levels, the second with 2^3=8
#levels.
#fraction = Full design: fraction=0, half-fraction:
#fraction=1 etc.
#gen = list of generators at bit-factor level, e.g.
#list(c(1,4)) is the interaction between bit-factor
#1 and 4.
#design = list of the order of the design, e.g.
#c(1:4,7:9,13:15,19:21,24:26,5,6,10:12,16:18,22,23
#,27,28)
#It specifies the order in which to use the original
#bit factors.

#Outputs
#BitDesign - containing the bit design of the
#design problem
#Design - containing the replaced numerical design
#usable on the design factors.
```

```
mbrdLarge <- function(l2levels=c(2,2), fraction=0,
                      gen=NULL, design=NULL...){
    require(FrF2)
    nfac <- length(l2levels)
    fnames1 <- character()
    for(i in 1:nfac){
        fnames1 <- c(fnames1,paste(letters[i],
                                   1:l2levels[i],
                                   sep=""))
    }
    fnames2 <- Letters[1:nfac]

    #This additional code checks if an order is given
    #and sets the names in desired order.
    #The design.order is then passen on to the call in
    #FrF2Large, to use as column names for the bit
    #design created there.
    if (is.null(design)){
      design.order <- fnames1
    } else {
      design.order <- fnames1[design]
    }


    D1 <- FrF2Large(nfactors=sum(l2levels),
                    randomize=FALSE,
                    nruns=2^(sum(l2levels)-fraction),
                    generators=gen,
                    factor.names=design.order)
    D2 <- ifelse(D1==-1,0,1)

    #The returned designs columns are reordered into the
    #original order of the factors,before it is turned
    #back into int-values instead of bits.
    org.order <- D2[,order(design.order)]

    D3 <- t(apply(org.order,1,bits2int,
                  l2levels=l2levels))

    colnames(D3) <- fnames2
    res <- list(BitDesign = D1, Design = D3)
```

```
    return(res)
}

#Utility function for mbrd for translating bits
#to integers
bits2int <- function(x, l2levels){
    require(sfsmisc)
    nfac <- length(l2levels)
    cumlevels<-c(0,cumsum(l2levels))+1
    intvec <- rep(0,nfac)
    for(i in 1:nfac){
        z <- x[cumlevels[i]:(cumlevels[i+1]-1)]
        intvec[i] <- polyn.eval(z, 2)
    }
    return(intvec)
}
```

Listing B.2: This code contains the function to validate the design and parameter values

```
# ------------ Function valid.design() ----------------
# Taking in a design of runs

valid.design <- function(run, l2levels = c(7,8,8,5,5)){

  n.runs <- dim(run)[1]

  x <- round(seq(0.001,to=1,length=100),digits=5)

  e <- round(seq(0,0.5,length=2**l2levels[1]),
             digits=5)
  mu1 <- round(seq(0.01,1,length=2**l2levels[2]),
               digits=5)
  mu2 <- round(seq(0.01,1,length=2**l2levels[3]),
               digits=5)
  sd1 <- round(seq(0.001,1/6,length=2**l2levels[4]),
               digits=5)
  sd2 <- round(seq(0.001,1/6,length=2**l2levels[5]),
               digits=5)

  ind <- logical(0)
  j <- 1
  for (i in c(1:n.runs)){
    ei <- e[run[i,1]+1]
```

```
    if ((sd1[run[i,4]+1] < min((mu1[run[i,2]+1]/3),
                               (1-mu1[run[i,2]+1])/3))
        && (sd2[run[i,5]+1] <
          min((mu2[run[i,3]+1]/3),
              (1-mu2[run[i,3]+1])/3)))
      {
      ind[i] <- TRUE
      j <- j + 1
    }else{
      ind[i] <- FALSE
    }
  }
  n <- j - 1

  small.ind <- which(ind)
  return(list("index"=small.ind,"n"=n))
}
```

Listing B.3: This code contains the function to generate the database

```
# --- Function generate.F() ---
# Takes in a run scheme (design) found using mbrd,
# customizable levels, and a filename to save it in.
# Default number of level are consistent with the
# rest of the implementation, but makes it easier
# to switch the number of levels.

generate.F.prcomp <-
  function(run, l2levels = c(8,9,9,7,7),
           filename = "default_curve_lib.rdata"){
  # --- Initializing the parameter values ---

  x <- round(seq(0.001,to=1,length=100),digits=5)
  e <- round(seq(0,0.5,length=2**l2levels[1]),
             digits=5)
  mu1 <- round(seq(0.01,1,length=2**l2levels[2]),
               digits=5)
  mu2 <- round(seq(0.01,1,length=2**l2levels[3]),
               digits=5)
  sd1 <- round(seq(0.001,1/6,length=2**l2levels[4]),
               digits=5)
  sd2 <- round(seq(0.001,1/6,length=2**l2levels[5]),
               digits=5)
```

```
# --- Finding a valid set of lines to run ---

design <- valid.design(run,l2levels)
G <- run[design$index,]
n <- dim(G)[1]

# --- Finding function values ----

X <- matrix(data=0, ncol=100, nrow=n)
par.values <- matrix(data=0, ncol=5, nrow=n)
f1x <- matrix(data=0, ncol=100, nrow=n)
f2x <- matrix(data=0, ncol=100, nrow=n)

for (i in c(1:n)){
  ei <- e[G[i,1]+1]
  f1x[i,] <- pnorm(x,mean=mu1[G[i,2]+1],
                   sd=sd1[G[i,4]+1])
  f2x[i,] <- pnorm(x,mean=mu2[G[i,3]+1],
                   sd=sd2[G[i,5]+1])
  X[i,] <- ei*f1x[i,] + (1 - ei)*f2x[i,]
  par.values[i,] <- c(ei,mu1[G[i,2]+1],mu2[G[i,3]+1],
                      sd1[G[i,4]+1],sd2[G[i,5]+1])

}

ybar <- colMeans(X)

pres <- prcomp(X,retx=TRUE,center=TRUE)
scores <- pres$x
loadings <- pres$rotation
sdev<-pres$sdev
variance<-sdev*sdev
prop <- variance/sum(variance)
cum <- cumsum(prop)
antall <- min(which(cum>0.999))

# Selecting this number of scores
scores <- scores[,1:antall]
loadings <- loadings[,1:antall]

residuals <- matrix(0, ncol=100, nrow=n)
errors <- rep(0,n)
for (j in 1:n){
```

```
    residuals[j,] <-
      (X[j,]-ybar) - scores[j,]%*%t(loadings)
    errors[j] <-
      (t(residuals[j,])%*%(residuals[j,]))/100
  }

  # These are the limits for the errors
  limits <- quantile(errors, probs=c(.01,.05,.95,.99))

  center <- pres$center
  # ------------- Saving and returning ---------------
  save(G, x, loadings, scores, center, ybar, limits, n,
       par.values, file=filename)
  return(list("G"=G, "x"=x, "loadings"=loadings,
              "scores"=scores, "center"=center,
              "ybar"=ybar, "limits"=limits,
              "n" = n, "par.values"=par.values))
}
```

Listing B.4: This code contains the function to process new observations before performing look-up

```
# --- Process data observations and return
# estimated distribution line and offsets
# and slopes ---
process.observations <- function(obs){
  d <- density(obs, n=100)
  dx <- d$x
  off.x <- dx[1]
  sl.x <- dx[100]-dx[1]
  new.x <- (x*sl.x - 0.001)/(1-0.001) + off.x
  dy <- d$y
  cum <- cumsum(dy)
  off <- cum[1]
  sl <- cum[100] - cum[1]
  st.obs <- (cum - off)/sl

  return(list("st.obs"=st.obs,"new.x"=new.x,
              "off"=off,"sl"=sl,
              "off.x"=off.x,"sl.x"=sl.x))
}
```

Listing B.5: This code contains the function to perform look-up of a new variable

```
# --- Function DLU ---
# input: y to look up, the loadings, scores, ybar
# and parameter values if a curve library
# output: the function parameter, the errors and the
# distances of the 10 best lines

dlu <- function(y, v, scores, ybar, par.values){

  yv <- (y - ybar)%*%v

  n <- dim(scores)[1]

  distance <- rep(0,n)
  for (i in c(1:n)){
    distance[i] <- sqrt((yv - scores[i,])%*%
      t(yv - scores[i,]))
  }

  order.distance.ind <- order(distance)[1:100]
  distance.small <- distance[order.distance.ind]

  scores.small <- scores[order.distance.ind,]

  residual <- matrix(ncol=100,
                     nrow=length(distance.small))
  error <- rep(0,length(distance.small))

  for (i in c(1:length(distance.small))){
    residual[i,] <- (y - ybar) -
      scores.small[i,]%*%t(v)
    error[i] <- (t(residual[i,])%*%
      (residual[i,]))/100
  }

  least.ind <- order(error)[1:10]
  least.error <- error[least.ind]
  least.distance <- distance.small[least.ind]

  func.par <-
    par.values[order.distance.ind[least.ind],]
```

```
  return(list("par.org" = func.par,
              "error" = least.error,
              "distances"=least.distance))
}
```

Listing B.6: This code contains the function to process the look-up parameters back to original x space

```
back.processing <- function(par, sl, off){

  par.back <- matrix(0,ncol=5,nrow=10)
  par.back[,1:2] <- (par[,2:3]*sl+0.001)/(.999)+off
  par.back[,3:4] <- par[,4:5]*sl/(.999)
  par.back[,5] <- par[,1]

  return(par.back)
}
```

B.2 Performance

The method performance is important, and all estimations and predictions were run on simulated data from random distributions.

Listing B.7: Code for sample generation used to generate all samples that have been used in evaluation of generators and methods

```
n.list <- c(50,100,200,500)
k <- 1
for ( i in 1:4){
  print(paste("i: ",i))
  nsim <- n.list[i]
  for (j in 1:200){
    print(paste("j: ",j))

    eps[k] <- runif(1,0,1)
    mu1[k] <- runif(1,-3,3)
    mu2[k] <- runif(1,-3,3)
    s1[k] <- runif(1,0.01,3)
    s2[k] <- runif(1,0.01,3)

    n1[k] <- rbinom(1, nsim, eps[k])
    n2[k] <- nsim-n1[k]
    Xobs[k,1:nsim] <- c(rnorm(n1[k], mu1[k], s1[k]),
                        rnorm(n2[k], mu2[k], s2[k]))
```

```
    XTEST[k,1:nsim] <- c(rnorm(n1[k], mu1[k], s1[k]),
                         rnorm(n2[k], mu2[k], s2[k]))

    k <- k + 1
  }
}
```

Listing B.8: This code contains a minimum working example of the above code and describes how the code was used to estimate the problem

```
# --- Minimum working example,
# demonstrate how files are used ---

source("mbrd_large.R")
source("dlu_source.R")
source("valid_design.R")
source("process_observations_source.R")
source("backprocess_source.R")
source("generateF_prcomp_center.R")

# --- find set of generators
gen.list <- list(0)
bits <- 1:16
gen <- replicate(24, sample(bits, 7))
for (z in 1:24){
  gen.list[[z]] <- c(gen[,z])
}
des <- c(1:4,9:12,18:21,27:28,34:35,5:8,
         13:17,22:26,29:33,36:40)

# --- Generate design and function data base
res <- mbrdLarge(c(8,9,9,7,7), fraction=24,
                 gen=gen.list, des)
run <- res$Design
F <- generate.F.prcomp(run, c(8,9,9,7,7),
                       filename="mwe_test.rdata")

# --- Example mixture to estimate
eps <- 0.3
mu1 <- 0
mu2 <- 2
s1 <- 0.5
s2 <- 1
N <- 50
```

```
truepar <- c(mu1, mu2, s1, s2, eps)

# --- simulating data and test data from mixture ---
n1 <- rbinom(1, N, eps)
n2 <- N-n1
X <- c(rnorm(n1, mu1, s1), rnorm(n2, mu2, s2))
TESTX <- c(rnorm(n1, mu1, s1), rnorm(n2, mu2, s2))

st <- process.observations(X)

lookup <- dlu(st$st.obs, F$loadings, F$scores,
              F$ybar, F$par.values)

back.par <- back.processing(lookup$par.org,
                            st$sl.x,st$off.x)

plot.parameter.estimates(truepar,st$new.x,col="red")
points.parameter.estimates(back.par[1,],st$new.x)
points.parameter.estimates(back.par[2,],st$new.x)
points.parameter.estimates(back.par[3,],st$new.x)
lines(st$new.x,density(X,n=100)$y,col="forestgreen")

plot.parameter.densities(truepar,st$new.x,col="red")
points.parameter.densities(back.par[1,],st$new.x)
points.parameter.densities(back.par[2,],st$new.x)
points.parameter.densities(back.par[3,],st$new.x)
lines(st$new.x,cumsum(density(X,n=100)$y)/st$sl,
      col="forestgreen")

restab <- cbind(truepar,t(back.par))
logliks <- apply(restab,2,loglik,x=TESTX)

loglikdiff <- logliks - logliks[1]
restab <- rbind(restab,logliks, loglikdiff)
dimnames(restab) <- list(c("mu1","mu2", "sigma1",
                           "sigma2","eps", "loglik",
                           "diff"),
                         c("True","DLU1","DLU2","DLU3",
                           "DLU4","DLU5","DLU6","DLU7",
                           "DLU8","DLU9","DLU10"))

print(restab)
```

```
# --- Functions for plots and log likelihood ---

plot.parameter.estimates <-
  function(p=c(mu1,mu2,s1,s2,eps),x,
           col="black",ylim=NULL){
  f <- p[5]*dnorm(x,p[1],p[3]) +
    (1-p[5])*dnorm(x,p[2],p[4])
  plot(x,f,type="l",col=col,ylim=ylim)
}
points.parameter.estimates <-
  function(p=c(mu1,mu2,s1,s2,eps),x,
           col="black"){
  f <- p[5]*dnorm(x,p[1],p[3]) +
    (1-p[5])*dnorm(x,p[2],p[4])
  points(x,f,type="l",col=col)
}

plot.parameter.densities <-
  function(p=c(mu1,mu2,s1,s2,eps),x,
           col="black",ylim=NULL){
  f <- (p[5]*pnorm(x,p[1],p[3]) +
    (1-p[5])*pnorm(x,p[2],p[4]))
  plot(x,f,type="l",col=col)
}
points.parameter.densities <-
  function(p=c(mu1,mu2,s1,s2,eps),x,
           col="black"){
  f <- (p[5]*pnorm(x,p[1],p[3]) +
    (1-p[5])*pnorm(x,p[2],p[4]))
  points(x,f,type="l",col=col)
}

loglik <- function(pars,x){
  likvec <- pars[5]*dnorm(x, pars[1], pars[3]) +
    (1-pars[5])*dnorm(x, pars[2], pars[4])
  loglik <- sum(log(likvec))
  return(loglik)
}
```