Number of component selection in finite mixture models

Testing the robustness of BIC-based estimation, and possible improvement using AIC

by

J.Y.P. Wong

to obtain the degree of Bachelor of Applied Mathematics
at the Delft University of Technology,
to be defended publicly on Wednesday July 8 at 13:30.

Student number:	5643910	
Project duration:	April 20, 2026 – July 8, 2026	
Thesis committee:	Dr. Ö. Şahin,	TU Delft, supervisor
	Dr. ir. J. Bierkens,	TU Delft

An electronic version of this thesis is available at `http://repository.tudelft.nl/`.

TU Delft

Acknowledgements & Declarations

I would like to thank my supervisor, Professor Özge Şahin, for her guidance and valuable feedback throughout the writing process of this thesis.

In the processes of assisting with literature search, language editing and assisting in code writing, generative AI (Claude, Anthropic, 2026) was used during the writing of this thesis. That being said, all output of such AI was reviewed and verified by the author, instead of being used directly.

Abstract

Multivariate Gaussian mixture models commonly use the Bayesian Information Criterion (BIC) in order to estimate the number of components in a dataset. Partially inspired by the decomposition of the covariance matrix of bivariate Gaussian distributions by Banfield and Raftery (1993), experiments were carried out testing whether this model selection criterion performs well in estimating the correct number of components and model type over a range of values for the horizontal mean of the mean vector, orientation-related parameter and volume-related parameter. In Chapter 3, it was often found that BIC-based selection estimated the wrong number of components when the generative distributions of the 2 components were nearly identical. This similarity between the generative distributions resulted in consistent underestimation of the number of components using BIC. In an attempt to improve estimations, the Akaike Information Criterion (AIC) was also considered and used for the experiments of the horizontal mean and volume experiments, as it punishes complex models less severely. In both settings, AIC gave better results on average for the estimated number of components for the parameter values where BIC-based estimation generally performed poorly. The downside of AIC was regular overestimation of the number of components, which came from its aforementioned leniency. Regardless, number of component estimation using AIC did have areas of better performance than BIC. An experiment using copulas for the generation of paired data was performed in a similar setting to the horizontal mean experiment, but both BIC and AIC showed bad results that were often incorrect in terms of estimating the true number of components.

Layman's Summary

In data analysis, it is possible for data to come from multiple sources. When working with such data, it may be useful to estimate how many sources generated it. Methods to estimate this exist, but they rely on certain assumptions. It is tested how well these methods work when the sources are made very similar to one another, which would make it more difficult to separate different sources. Performing such experiments, it was found that indeed the existing, commonly used methods struggle when faced with such data from sources that are highly similar. A different method was introduced, which estimates the amount of sources more accurately than the existing methods where the latter struggles. Its drawback is that it performs worse in situations where the existing method already identifies the correct number of sources well. Regardless, both the existing as well as the new method introduced are effective and have their advantages. Finally, sources are considered with structures that have shapes that the existing method was not designed for. Unfortunately, both the existing as well as the new method perform quite badly due to the irregular structures of these sources.

Contents

1

Introduction

Data is everywhere in the current world, and data analysis comes with it naturally. However, there are plenty of times where the results of data analysis turn out to be inconclusive or yield counterintuitive results. Although there may be an infinite number of reasons for one of these scenarios to occur, one of those reasons may be helped by mixture models.

What sets mixture models apart from regular statistical models (often encountered in introductory statistics classes), is that it was often assumed that data came from a single distribution. The task would then often be to estimate the type of distribution and parameter values of *the* underlying distribution. Although simple, this assumption may restrict the domain of models too much for the data at hand, which may instead have come from multiple underlying subgroups. These subgroups, formally called components, can each have their own distribution type and parameter values. If these components were to be identified and analysed from a set of data, then conclusions could be taken from these components separately. Not separating data into components (properly) can create a well-known phenomenon known as Simpson's paradox (Simpson 1951). For a practical example of this phenomenon, consider following results of success rates for 2 different treatments for Kidney stones in Table 1.1. This data is taken from Charig et al. (1986).

	Treatment A	Treatment B
All data	78% (273/350)	83% (289/350)
Small kidney stones	93% (81/87)	87% (234/270)
Large kidney stones	73% (192/263)	69% (55/80)

Table 1.1: Table of example success rates for kidney stone treatments. In brackets, the number of succesful and total treatments for each group are shown respectively.

As can be seen, treatment B has a higher success rate when considering all recorded cases. However, after identifying subgroups (in this example the size of the kidney stones) and analyzing them separately, we find that treatment A is actually more effective in each of the subgroups. This different outcome arises due to the patients not being divided evenly over the 2 treatments when considering the subgroups.

We thus see that in this example, a different conclusion would have been found (that seems like an incorrect conclusion) if we never considered that the data consists of subgroups.

Research has already been done on the topic of mixture models, and estimating various aspects of such mixture models like the most likely number of components has also been set up extensively, for example in Celeux and Govaert (1995). In particular, Gaussian mixture models are often considered. These mixture models simplify the idea of mixture models slightly as they assume each component to have a Gaussian distribution (Banfield and Raftery (1993)). The `mclust` package of the R programming language is commonly used to estimate exactly this number of components given a data set (Scrucca, Fop, et al. (2016)). In this thesis, I will be aiming to answer the following:

Research Question:
How robust is Bayesian Information Criterion (BIC) (Schwarz (1978))-based number of component selection in finite Gaussian mixture models to violations of its underlying assumptions, and can Akaike Information Criterion (AIC) (Akaike (1974))-based number of component selection produce better performance?

As one might guess from the research question, BIC and AIC are used in estimating the best model, and will be introduced later in Section 2.5.2 and Section 4.1 respectively. Furthermore, finite Gaussian mixture models are simply Gaussian mixture models with a finite number of components. These selection methods have been compared using simulation experiments in which 2-component Gaussian mixture models were generated that violated the assumptions of BIC-based selection. Specifically, varying horizontal mean, orientation and volume parameters of the generative components were the central point of the experiments. These generative components ranged from clearly separated to nearly identical, where `mclust` was used to estimate the preferred model type and number of components based on BIC, and this was repeated using AIC as well which has a penalty term that punishes complex models less. Additionally, copula-based mixture models were created and experimented with to test how performance was affected when faced with the resulting non-elliptical dependence structures. When compared, we find that BIC tends to underestimate the true number of components, while AIC overestimates it. In addition to this, where one tends to be inaccurate, the other tends to be more accurate.

In this thesis, programming will be used for plenty of different purposes, including but not limited to:

- Visualizing data
- Comparing the ability of different models to represent given data
- Selecting the model that best fits given data

For these reasons, the programming language R will be used.

In Chapter 2, mixture models are introduced (Banfield and Raftery 1993) , motivated and it is explained how the most suitable type of mixture model among different types is selected. Using this knowledge, in Chapter 3 experiments are performed testing the robustness of BIC-based number of components selection. Following the results of these experiments, we attempt to use AIC-based selection in Chapter 4 in order to find different results for these same experiments, while also considering copulas for the generation of bivariate data (Nelsen 2006). Finally, Chapter 5 concludes by answering the research question and suggesting different possible ideas for future work.

2

Mixture Models

2.1. What are mixture models?

The concept of mixture models is an extension on the core assumption made in (introductory) statistical courses, that being that the n independent and identically distributed (often abbreviated to iid) (possibly multivariate) points of data $\mathbf{x} = (\mathbf{x}_1, \ldots, \mathbf{x}_n)$ come from a *single* underlying distribution. The goal is then to estimate the type of distribution and its parameter values. Mixture models remove this assumption, assuming instead that each independent observation $\mathbf{x_i}$ comes from *one of the* underlying distributions, or intuitively from one of the G groups, formally called *components*. Therefore, mixture models instead assume a heterogeneous population. Some important details to note here are:

- Not all groups may be represented equally within the population. For this reason, we will introduce notation τ_k for the probability that any observation belongs to the kth component.
- Similarly to general probabilities, the component probabilities $\tau_1, ..., \tau_G$ should satisfy:
  - $\forall k : \tau_k \geq 0$
  - $\sum_{k=1}^{G} \tau_k = 1$
- Although it is often assumed for simplicity that all components follow the same type of distribution (thus only differing in parameter values), multiple types of distribution may be present among the G components. That being said, in almost all of this paper we assume the same type of distribution among all components, in particular the bivariate Gaussian distribution.

Let us use for the kth component the notation $\boldsymbol{\theta}_k$ for its parameter values and f_k its density function, then the likelihood for a mixture model with G components is

$$\mathscr{L}_{MIX}(\boldsymbol{\theta}_1, ..., \boldsymbol{\theta}_G; \tau_1, ..., \tau_G | \mathbf{x}) = \prod_{i=1}^{n} \sum_{k=1}^{G} \tau_k f_k(\mathbf{x}_i | \boldsymbol{\theta}_k). \tag{2.1}$$

2.2. Why are mixture models important?

The single-distribution assumption of regular statistical modelling, although avoiding a lot of additional work, is not always accurate in practice. To illustrate this more clearly, let us consider the following histogram in Figure 2.1 containing example data about the travel times in minutes for 1000 people travelling from the same starting point to the same destination.

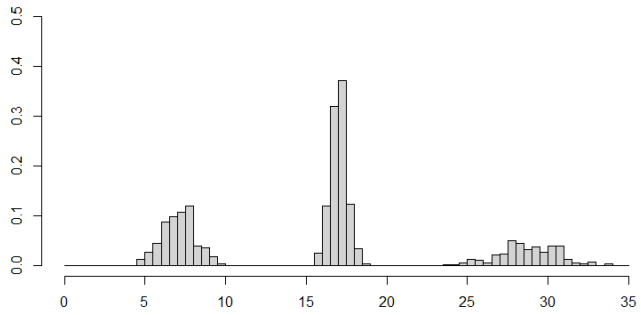

Figure 2.1: Example data representing travel times in minutes.

If we restrict our options to only consider univariate Gaussian distributions for a moment, it would not be too far-fetched to think that the data consists of 3 Gaussian components. Let us now compare the models estimated using `mclust` that assume 1 and 3 components. This `mclust` package is commonly used to fit and compare Gaussian mixture models across a range of possible component numbers and model types. How this best model is selected will be discussed in Section 2.5, and more information about this package can be found Scrucca, Fraley, et al. (2023).

Estimated parameter	$G = 1$	$G = 3$
Means	1.39	(7.08, 17.04, 28.78)
Variances	57.95	(1.05, 0.25, 3.56)
Component proportions	1	(0.30, 0.50, 0.20)

Table 2.1: Comparison estimated 1- and 3-component Gaussian models for the mentioned example travel time data.

Although `mclust` already makes use of a model selection criterion, formal criteria for model selection will be discussed extensively later (specifically in Section 2.5.2). Regardless, fitting these 1- and 3-component models using `mclust` produces Figure 2.2. The red and blue curves correspond to the 1- and 3-component models respectively, and visually it does not seem like too much of an outlandish conclusion to say that the mixture model is a better fit.

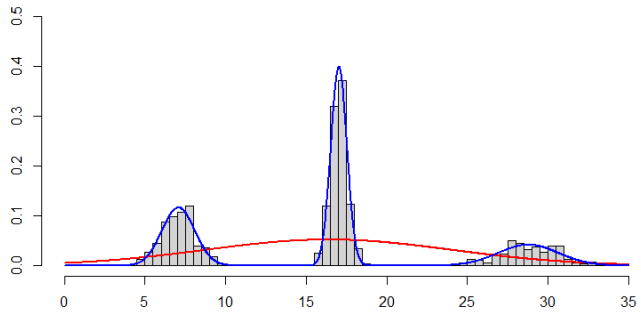

Figure 2.2: Plot with fitted models curves for the travel time example data.

Scaling up to 2-dimensional data, let us consider another example. This time, Figure 2.3 represents data of measured heights and weights of 300 animals.

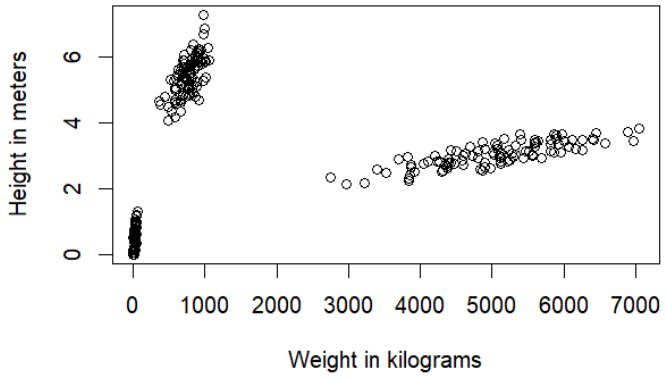

Figure 2.3: Example data representing animal heights and weights.

Again, it seems visually like the data would consist of 3 groups. More specifically, attempting to analyze each of the groups gives us the following preliminary descriptions of entries in said groups:

1. (In the bottom left) Animals that are (relatively) light and small like dogs.
2. (In the top left) Animals that are very tall, yet not too heavy like giraffes.
3. (On the right) Animals that are decently tall, but more significantly relatively very heavy like elephants.

We now have already been able to draw some distinctive conclusions about the different components, even before any formal analysis has been performed. Comparing the estimated Gaussian mixture models for 1 and 3 components using `mclust` once more, we now get the following (values rounded for brevity):

Estimated parameter	$G = 1$	$G = 3$
Means (Height, Weight)	(2.99, 1926.26)	(0.58, 27.45), (5.45, 792.92), (2.94, 4958.41)
Component proportions	1	0.3333, 0.3333, 0.3333

Table 2.2: Comparison of 1- and 3-component Gaussian models for animal data.

The corresponding estimated covariance matrices for the $G = 1$ model is

$$\hat{\Sigma} = \begin{bmatrix} 4.14 & 663.19 \\ 663.19 & 4932845.81 \end{bmatrix},$$

while for the $G = 3$ model the estimated covariance matrices for the three components are

$$\hat{\Sigma}_1 = \begin{bmatrix} 0.06 & 1.61 \\ 1.61 & 123.03 \end{bmatrix}, \quad \hat{\Sigma}_2 = \begin{bmatrix} 0.31 & 40.54 \\ 40.54 & 17491.59 \end{bmatrix}, \quad \hat{\Sigma}_3 = \begin{bmatrix} 0.18 & 312.94 \\ 312.94 & 697064.01 \end{bmatrix}.$$

The overlays of the fitted distributions of the 1- and 3-component models are then as in Figure 2.4 and Figure 2.5, respectively. Once again, it seems visually clear that the mixture model is the better choice here.

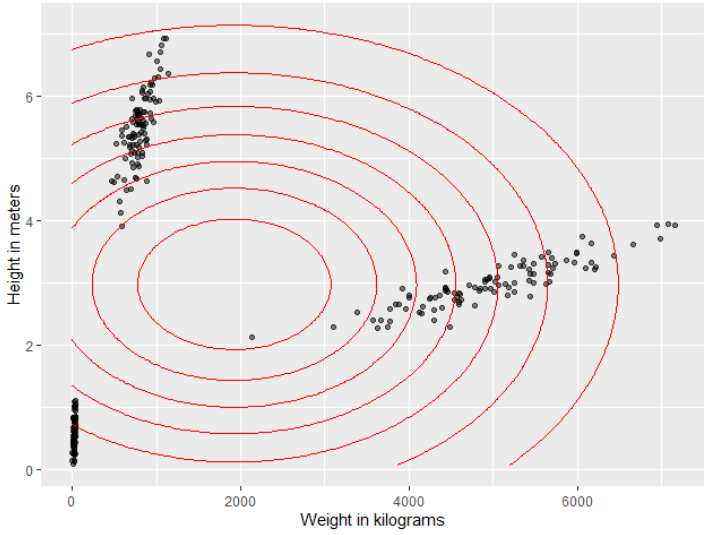

Figure 2.4: Plot of data with fitted single-component model curve for mentioned example animal data in Figure 2.3.

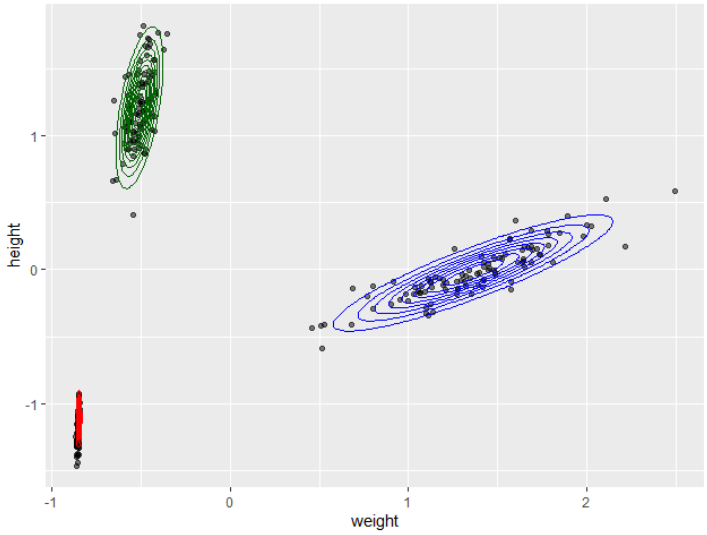

Figure 2.5: Plot of data with fitted multi-component model curves for mentioned example animal data in Figure 2.3.

It seems like a good idea to consider the possibility of data being composed of multiple different distributions, rather than just a single distribution. That being said, it is obviously more difficult to estimate parameters for a mixture of multiple distributions rather than a single distribution. The R code used to create all figures in this chapter can be found in Appendix B.

2.3. Mixture models parameters

Now that we have seen why mixture models are important to consider, we look a bit more into the fitting of mixture models. More specifically, we will look at the types of mixture models to choose from when considering mixture models where all components follow a multivariate Gaussian distribution (usually bivariate). Under this assumption, the type of distribution will not have to be estimated for each of the multiple components.

2.3.1. Multivariate Gaussian distribution

As it is assumed for each component to follow the multivariate Gaussian distribution, let us look into the parameters of this distribution first. The multivariate Gaussian distribution is commonly denoted by $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$,

and contains the following parameters:

- The mean vector $\boldsymbol{\mu}$ which contains the mean of each variable.
- The covariance matrix $\boldsymbol{\Sigma}$ which contains both the variance of the variables as well as the covariances between pairs of different variables.

In case of d-dimensional data, this leads to dimensions $\boldsymbol{\mu} \in \mathbb{R}^d$ and $\boldsymbol{\Sigma} \in \mathbb{R}^{d\times d}$ for the mean vector and covariance matrix respectively. Now it is important to note that, although there are no restrictions for the mean vector $\boldsymbol{\mu}$, not any matrix is a valid covariance matrix. For the sake of explanation, let us consider 2-dimensional data and in turn a corresponding 2x2 covariance matrix $\boldsymbol{\Sigma}$ (often denoted as $\boldsymbol{\Sigma}_k$ in case there is merit to mentioning the dimension explicitly). For a valid covariance matrix $\boldsymbol{\Sigma}$, we must have that:

- $\boldsymbol{\Sigma}$ is symmetric, as the non-diagonal entries of a covariance matrix represent the covariances. Symmetry is then implied as $\text{Cov}(X_i, X_j) = \text{Cov}(X_j, X_i)$.
- $\boldsymbol{\Sigma}$ has non-negative diagonal entries, as these entries represent variances. As for any random variable X we must have $\text{Var}(X) \geq 0$, this condition is implied.
- For the determinant of $\boldsymbol{\Sigma}$ we have $\det(\boldsymbol{\Sigma}) \geq 0$. This condition is less immediately obvious than the others, but a consequence of this condition is the fact that for correlation coefficient ρ, we have $-1 \leq \rho \leq 1$.

The probability density function of a multivariate Gaussian distribution, given valid $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$, is defined as

$$f(\mathbf{x}) = (2\pi)^{-k/2} \det(\boldsymbol{\Sigma})^{-1/2} \exp\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right). \tag{2.2}$$

2.3.2. General mixture models

Now when a model consisting of G Gaussian components is chosen, the parameters that are to be estimated are not limited to the mean vectors $\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, ..., \boldsymbol{\mu}_G$ and covariance matrices $\boldsymbol{\Sigma}^1, \boldsymbol{\Sigma}^2, ..., \boldsymbol{\Sigma}^G$ of the multivariate Gaussian distributions. In addition to these parameters, the component probabilities $\tau_1, ..., \tau_G$ of each class should also be estimated. Notice moreover that a superscript is used for component covariance matrices to avoid confusion with the dimensions of the covariance matrix $\boldsymbol{\Sigma}_k$.

2.4. Parameter estimation in multivariate Gaussian mixture models

When fitting a model of G Gaussian components to a given data set containing d-dimensional data, the parameters that were mentioned in Section 2.3 need to be estimated. Let us consider each aspect of the mixture model, along with the number of parameters that require estimation:

- Mean vectors $\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, ..., \boldsymbol{\mu}_G$: For each dimension of the data, the mean must be estimated. Thus resulting in k parameters per component and $k * G$ parameters in total.
- $\boldsymbol{\Sigma}^1, \boldsymbol{\Sigma}^2, ..., \boldsymbol{\Sigma}^G$: As each $k \times k$ covariance matrix must be symmetric, the number of parameters to be estimated for each component is equal to the amount of entries on and above the diagonal $1+2+...+k = k(k+1)/2$, where the fraction follows from the Gauss sum. In total, we find $G(k(k+1)/2)$ parameters.
- Component probabilities $\tau_1, \tau_2, ..., \tau_G$: The number of component probabilities is trivially only dependent on the number of components in the model, making for a total of G parameters.

Comparing the three sources of parameters, it seems like the covariance matrices are responsible for the majority of the estimation requirements and in turn the computational work needed to fit mixture models to given data. In an attempt to lower the number of parameters required to be estimated for the covariance matrices, certain assumptions can be made about the distributions of each of the components. In order to explain these assumptions, we consider the following decomposition of a covariance matrix proposed by Banfield and Raftery (1993):

$$\Sigma_k = \lambda_k D_k A_k D_k^T. \tag{2.3}$$

In this decomposition, the factors indicate:

- λ_k: A constant, that is associated with the proportionality/volume of the component.

- D_k: An orthogonal matrix of eigenvectors, associated with the orientation of the component.
- A_k: A diagonal matrix with entries proportional to the eigenvalues, associated with the shape of the component.

Simplification of the mixture model can then be reached through the assumption that certain of these mentioned factors are the same across all components. Of course, the further these factors are constrained, the fewer parameters need to be estimated. Additionally, constraining the mixture model on purpose can prevent the component distributions from being "shaped" to the data too strictly, in turn overfitting to the data.

2.5. Model selection in multivariate Gaussian mixture models

Depending on the constraints enforced upon the mixture model, these can be divided into multiple different categories. These three types of models will each be covered in the following subsections, including the resulting form of the covariance matrices. Before this will be done however, it is important to note that different types of models are often condensed to a string of 3 letters, which indicates any constraints on the factors mentioned in Equation 2.3. More specifically, these letters represent (in order):

1. **Volume** of the component distributions, tied to λ_k.
2. **Shape** of the component distributions, tied to A_k.
3. **Orientation** of the component distributions, tied to D_k.

And these letters can be one of the following:

- **E**: Equal among all clusters.
- **V**: Allowed to vary across clusters.
- **I**: Equal to the identity matrix for all clusters. Note that this is only possible for the shape and orientation of a mixture model and thus not for the volume as it is a constant.

As an example, this means that model VEE represents a mixture model with components that have varying volumes, but equal shape and orientation. An example of data created from this model type is displayed below in Figure 2.6. This naming system is most notably used by the `mclust` package and thus also in plenty of places in the code used throughout this thesis. For more information on the different models that will be introduced in this section, see Celeux and Govaert (1995).

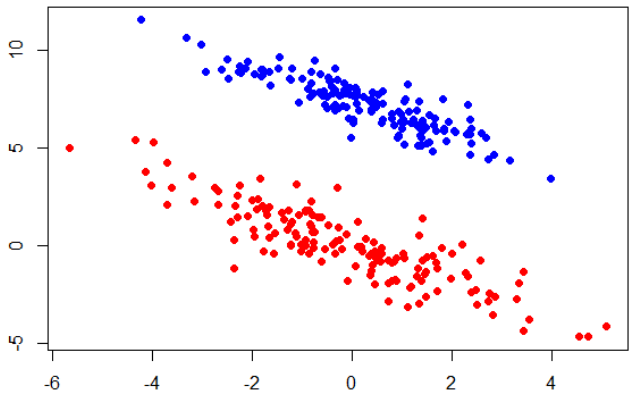

Figure 2.6: Example data corresponding to a VEE 2 model. Data observations from different components are represented with different colours.

2.5.1. Types of multivariate Gaussian mixture models

Ellipsoidal models

These most general types of models assume the non-restricted form of the covariance matrix as depicted in Equation 2.3, which results in the components of the fitted mixture models being able to obtain an ellipsoidal shape. Although this is the least restricted type of model among the ones to be covered in this section, assumptions of equal values of factors can still decrease the total number of parameters to estimate. This results in the following types of ellipsoidal mixture models:

- **VVV**: Varying volume, shape and orientation across all clusters.
- **VVE**: Varying volume and shape, but equal orientation across all clusters.
- **VEV**: Varying volume and orientation, but equal shape across all clusters.
- **EVV**: Varying shape and orientation, but equal volume across all clusters.
- **VEE**: Varying volume, but equal shape and orientation across all clusters.
- **EVE**: Varying shape, but equal volume and orientation across all clusters.
- **EEV**: Varying orientation, but equal volume and shape across all clusters.
- **EEE**: Equal volume, shape and orientation across all clusters.

In Figure 2.7, example generative distributions are shown for an ellipsoidal Gaussian mixture model, in particular the "VVV 3" model.

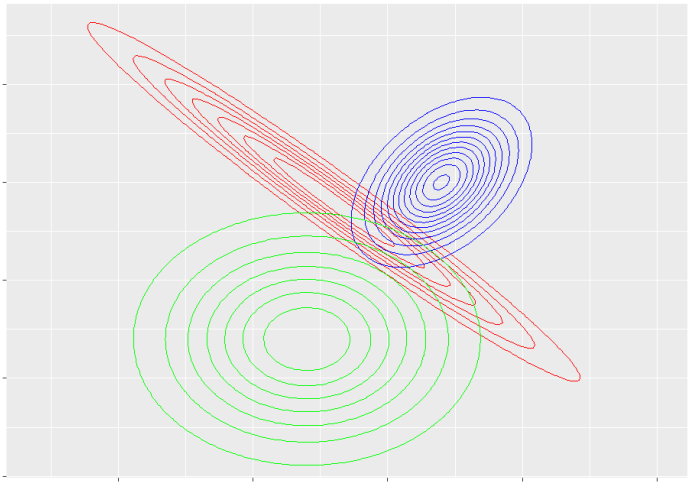

Figure 2.7: Example generative distributions for VVV 3 Gaussian mixture model type.

Diagonal models

Ellipsoidal models allow its components to have distributions that are oriented freely. Diagonal models in contrast do not allow such free orientation, by assuming that the orientation-related factor D_k is equal to the identity matrix. As a result, each component can only be aligned with the present coordinate axes. The covariance matrix of diagonal mixture models thus becomes

$$\Sigma_k = \lambda_k A_k \tag{2.4}$$

The different types of diagonal mixture models are as follows:

- **VVI**: Varying volume and shape across all clusters.
- **VEI**: Varying volume, but equal shape across all clusters.
- **EVI**: Varying shape, but equal volume across all clusters.

- **EEI**: Equal volume and shape across all clusters.

For this type of mixture models, the number of total parameters that require estimation has already decreased greatly. In Figure 2.8, example generative distributions are shown for a diagonal Gaussian mixture model, in particular the "EVI 3" model.

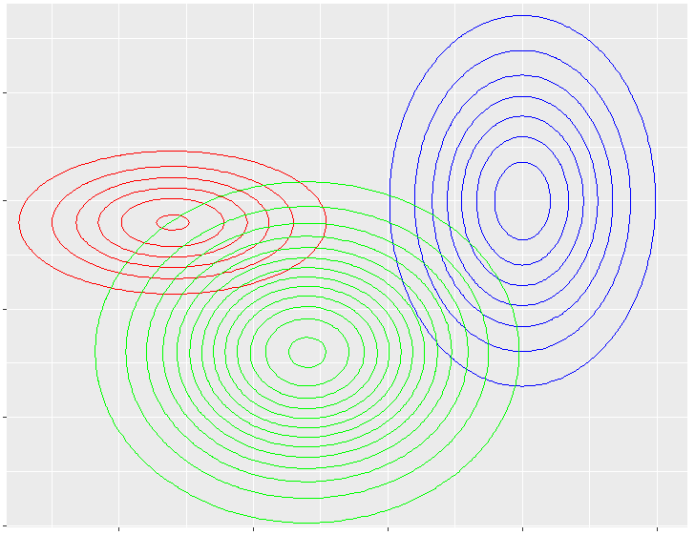

Figure 2.8: Example generative distributions for EVI 3 Gaussian mixture model type.

Spherical models
Although the property of free orientation is removed for diagonal models, the components of such models can still have differing shapes (that are orientated along the axes). As the name of this type of model may give away, the allowed oval shape of diagonal models can also be removed for further reduction of the number of covariance matrix parameters. This is done through the assumption that the shape-related factor A_k is also equal to the identity matrix, which results in the covariance matrix

$$\Sigma_k = \lambda_k I \tag{2.5}$$

The spherical mixture models only contain two different types:

- **VII**: Varying volume across all clusters.
- **EII**: Equal volume across all clusters.

In Figure 2.9, example generative distributions are shown for a spherical Gaussian mixture model, in particular the "VII 3" model.

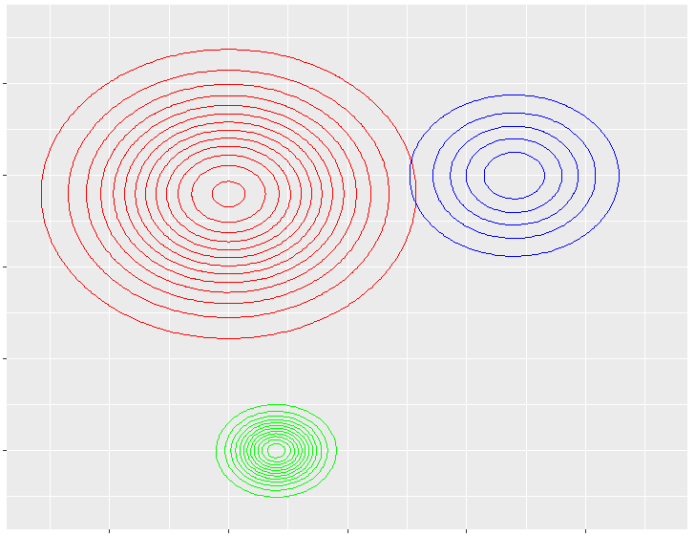

Figure 2.9: Example generative distributions for VII 3 Gaussian mixture model type.

By default, all of the fourteen associated model types are considered by the `mclust` package.

2.5.2. Model evaluation

Where the comparison between models was done solely through visual inspection in Section 2.2, the main method of model evaluation, and in turn model comparison, will be discussed now.

By default, `mclust` determines the best model type and number of components by taking each model-component combination (that was chosen to be considered) and calculating a quantity named the Bayesian Information Criterion. This quantity, commonly abbreviated to BIC, has the following Formula 2.6:

$$\text{BIC} = -2\ln(L) + k\ln(n), \tag{2.6}$$

where:

- L is the (maximised) likelihood for the considered model, thus representing how well the model fits the data.
- k is the number of parameters to be estimated for the considered model, thus representing the complexity of using said model.
- n is the number of data observations used.

If two models are compared, the one with a lower BIC is preferred. The $k\ln(n)$ term thus represents a penalty for complex models, as these bring with them a higher complexity. How well the model fits the data in the first place, represented in the $-2\ln(L)$ term, is still of great importance of course. There thus exists a trade-off: More complex models with no or fewer restrictions (as introduced in Section 2.5.1) likely require fewer components to fit given data accurately, than more simple models.

Although we will only simply compare models by determining which produces the smallest BIC value, it is suggested in Kass and Raftery (1995) that the absolute difference in BIC values when comparing different models (not necessarily Gaussian mixture models) can roughly be interpreted as in Table 2.3:

Δ_{BIC}	Evidence against model with higher BIC
Between 0 and 2	Negligible
Between 2 and 6	Significant
Between 6 and 10	Strong
Greater than 10	Very strong

Table 2.3: Rules of thumb for interpreting difference in BIC values as mentioned in Kass and Raftery 1995.

Now that BIC has been introduced, let us consider the different types of Gaussian mixture models discussed in Section 2.5.1 and see how the penalty term $k\ln(n)$ scales. In order to stay consistent with the experiments later introduced in Chapter 3, the number of parameters that need to be estimated k and the penalty term will be given for:

- Data set with 1000 observations ($n = 1000$).
- 2-dimensional data.

In the following Table 2.4, the number of parameters that need to be estimated and the resulting penalty term are given for the least restricted ellipsoidal, diagonal and spherical bivariate Gaussian mixture model types (VVV, VVI and VII respectively) for their 4-,3-,2- and 1-component models.

Model type	k	$k\ln(n)$
VVV 4	23	158.88
VVI 4	19	131.25
VII 4	15	103.62
VVV 3	17	117.43
VVI 3	14	96.71
VII 3	11	75.99
VVV 2	11	75.99
VVI 2	9	62.17
VII 2	7	48.35
VVV 1	5	34.54
VVI 1	4	27.63
VII 1	3	20.72

Table 2.4: Number of parameters to be estimated and penalty term BIC for different bivariate Gaussian mixture model types of 1- through 4-component bivariate Gaussian mixture models.

3

Number of component selection experiments

In this chapter, we will look to familiarise ourselves more with the different types of multivariate Gaussian mixture models. In particular, we will look at the behaviour of BIC-based model selection as implemented in the `mclust` package as it performs model selection on mixture models given data generated from specified component distributions. These component distributions, which will all be multivariate Gaussian distributions, will follow one of the multivariate Gaussian mixture model types mentioned in Subsection 2.5.1. It will then be analysed whether the correct type of model is still estimated using BIC-based model selection, as well as the estimated number of components. More specifically, these analyses will be performed as follows:

- **Model type**: The `mclust` package determines the BIC for each model-component combination (e.g. "EVE 3"), and stores all of these values. This allows for determining the ranking of each combination, with a rank of 1 indicating that the corresponding combination is the preferred model (as seen in Section 2.5.1, there are 14 model types for each number of components considered by `mclust` and for which the BIC is determined). The ranking of the known model-component combination is then determined and analysed in the results. The function for determining the rank of an arbitrary model type can be found in Appendix C.1.
- **Number of components**: In accordance with the rankings that were just mentioned, the number of components corresponding to a simulation is equal to the number of components in the model-component combination preferred using BIC-based model selection. For each experiment, only models with 1,2,3,4 or 5 components are considered by the BIC-based mode selection using `mclust` in order to reduce the time required by simulations. Models with even more components may still be considered, if the initial experiment results show a preference for models with 4 or 5 components.

The R code for any figures created in this chapter can be found in Appendix C.

3.1. 1D-mean shifting

In this experiment, we will consider data set containing data from 2 components following a bivariate Gaussian distribution with the exact same covariance matrix $\boldsymbol{\Sigma}$. The only differences in parameter values will thus be found in the mean vectors $\boldsymbol{\mu}_1, \boldsymbol{\mu}_2$, specifically by shifting 1 dimension of said mean vector. We thus aspire to find 2 components, and more specifically the model type "EEE 2". The specific parameter values used for this experiment (and other relevant details) are as follows:

- **Mean vectors** $\boldsymbol{\mu}_1 = (0,0)$ and $\boldsymbol{\mu}_2 = (x,0)$ with x variable with $x \in [-6,6]$.
- **Covariance matrices** $\Sigma_1 = \Sigma_2 = \begin{pmatrix} 1 & 0.5 \\ 0.5 & 1 \end{pmatrix}$.
- **Observations per component** $n_1 = n_2 = 500$.

- **Increment**: The value of the x-parameter increments by 0.1, i.e. $x_1 = -6, x_2 = -5.9, x_3 = -5.8, ..., x_{121} = 6$.
- **Simulations**: Per x-value, the experiment will be run 10 times (for creation of empirical result plots).

An example of generative distributions for this 1D-mean shift experiment can be seen in the following Figure 3.1.

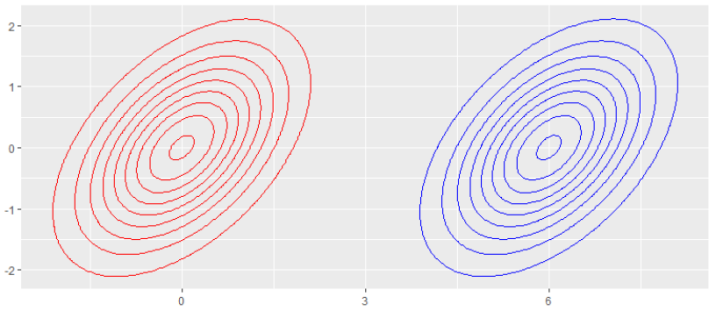

Figure 3.1: The generative distributions for 1D-mean shifting for $x = x_{121} = 6$.

Running the simulations for the 121 different values of x as described, we find results as shown in Figure 3.2. Here, the empirical mean of the "EEE 2" model ranking is shown, in addition to empirical standard deviation bands. Furthermore, a horizontal line is drawn where $y = 1$, indicating the desired rank of the model. Another line, now vertical, is also drawn at $x = 0$, as here both components have the exact same distribution parameters and effectively create a collection of data originating from 1 component.

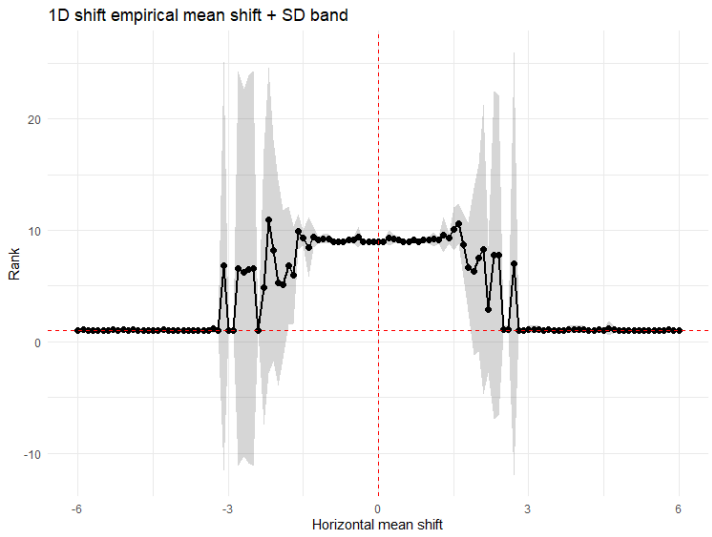

Figure 3.2: Plot of empirical mean with SD bands for ranking EEE 2 model. The empirical mean and SD bands are calculated over 10 simulations.

Upon visual inspection, it seems like the behaviour of the BIC-based selection can be divided into 3 sections:

1. Confident in correct model (Approximately where $|x| > 3$).
2. Other, incorrect models start to get favoured (Approximately where $1.5 > |x| \geq 3$).
3. Consistent in correct model not being the optimal model, with this model getting a consistent ranking of 9 (Approximately where $|x| \leq 1.5$).

We find that where $|x| \leq 3$, BIC-based model selection starts to consistently favour other models than the true "EEE 2" model. Thus we now consider a 100%-stacked bar chart of the fitted models in Figure 3.3.

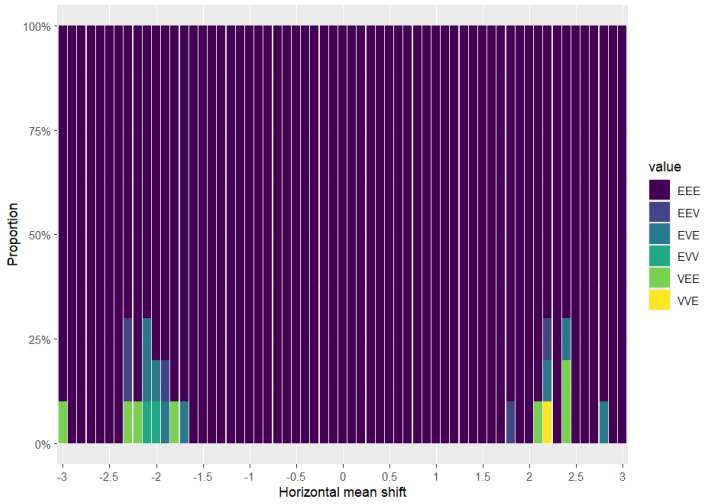

Figure 3.3: 100%-stacked bar chart of preferred model for 1D-mean shift experiment for $|x| \leq 3$. For each horizontal mean shift value, 10 simulations are performed.

Contrary to the results we just found in Figure 3.3, it seems like the correct model type is preferred almost always. This must then mean that the number of identified components of the fitted model is incorrect. For this reason, let us now consider another 100%-stacked bar chart in Figure 3.4, now for the fitted number of components over the same values of x.

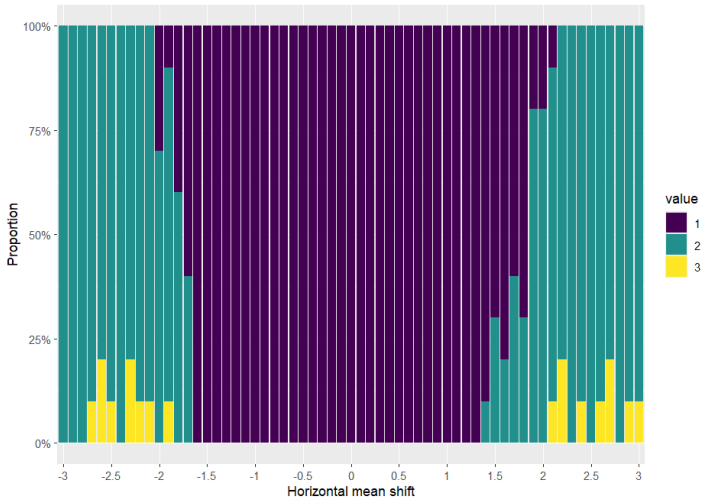

Figure 3.4: 100%-stacked bar chart of preferred number of components for 1D-mean shift experiment for $|x| \leq 3$. For each horizontal mean shift value, 10 simulations are performed.

Here it seems like indeed, more overlap between the generative distributions causes only 1 component to be identified. We thus find that a 1-component model is preferred as the absolute horizontal mean shift decreases. Figure 3.5 shows an example of data (and overlapping generative distributions) for which a single-component model was fitted incorrectly.

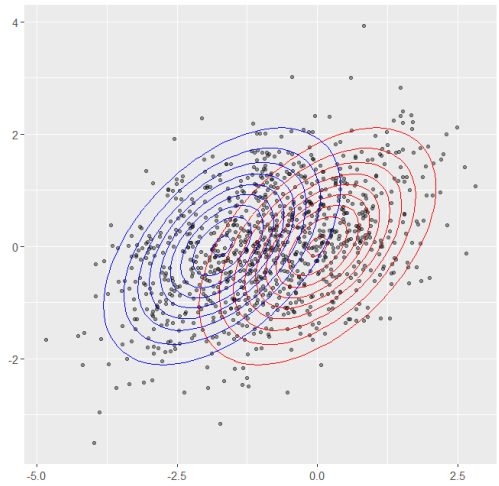

Figure 3.5: Data and generative distributions for $x = -1.7$. At this horizontal mean shift value, BIC-based model selection estimated the data given to come from a 1-component model in 6 out of 10 simulations.

3.2. Orientation shifting

In this experiment, we will consider a data set containing data from 2 components following a bivariate Gaussian distribution, this time with the exact same mean vector $\boldsymbol{\mu}$. What will differ now is the orientation-related factor D_k from the decomposition in Equation 2.3. Different orientations are considered using the rotational matrix $\begin{pmatrix} cos(\theta) & sin(\theta) \\ -sin(\theta) & cos(\theta) \end{pmatrix}$, which "rotates" the bivariate Gaussian distribution by an angle of θ radians. We thus aspire to find 2 components, and more specifically the model type "EEV 2". The specific parameter values used for this experiment (and other relevant details) are as follows:

- **Mean vectors** $\boldsymbol{\mu}_1 = \boldsymbol{\mu}_2 = (0,0)$.
- **Covariance matrices** $\Sigma_k = \lambda_k D_k A_k D_k^T$, where:
  - $\lambda_1 = \lambda_2 = 4$.
  - $A_1 = A_2 = \begin{pmatrix} 10 & 0 \\ 0 & 0.1 \end{pmatrix}$. It should be noted that the shape of both components is significant to the results of this experiment. For example, taking $A_1 = A_2 = I$ would create components with the same mean vector and a shape that is perfectly circular. In this scenario, changing the orientation should theoretically always lead to the components having the exact same probability density.
  - $D_1 = I$, $D_2(\theta) = \begin{pmatrix} cos(\theta) & -sin(\theta) \\ sin(\theta) & cos(\theta) \end{pmatrix}$ with θ variable with $\theta \in [0,\pi]$. Notice that, as the bivariate Gaussian distribution is symmetric among both axes (when disregarding orientations), we find that $\mathcal{N}(\boldsymbol{\mu}, \lambda_2 D_2(\theta) A_2 D_2^T(\theta))$ has the same probability density function (often shortened to pdf) as $\mathcal{N}(\boldsymbol{\mu}, \lambda_2 D_2(\theta+\pi) A_2 D_2^T(\theta+\pi))$ for any arbitrary value of θ.
- **Observations per component** $n_1 = n_2 = 500$.
- **Increment**: The value of the θ-parameter increments by $\pi/100$, i.e. $x_1 = 0, x_2 = \pi/100, x_3 = 2\pi/100, ..., x_{101} = \pi$.
- **Simulations**: Per θ-value, the experiment will be run 10 times (for creation of empirical result plots).

An example of generative distributions for this orientation shift experiment can be seen in the following Figure 3.6.

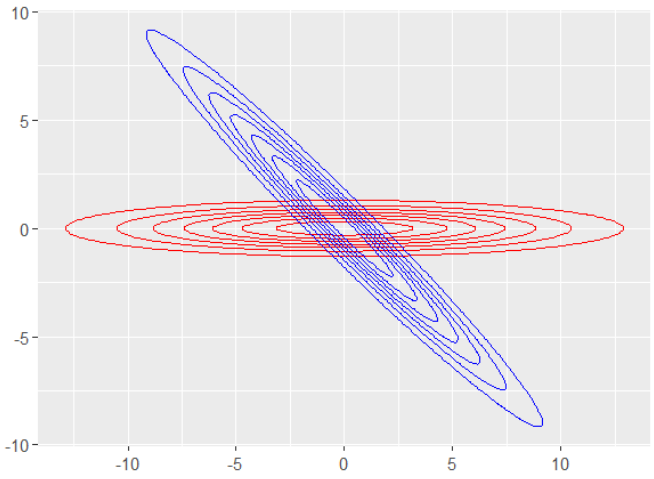

Figure 3.6: The generative distributions for orientation shifting for $\theta = \theta_{26} = pi/4$. For each orientation shift value, 10 simulations are performed.

Similarly to Section 3.1, we visualize the results of the described simulations in Figure 3.7 by plotting the empirical mean of the "EEV 2" model ranking and empirical standard deviation bands. Here, a horizontal line has again been added at desired model ranking 1. Different however, is the existence of 3 added vertical lines. The red lines, drawn at $\theta = 0$ and $\theta = \pi$, indicate places where the two components have the same pdf and the data thus effectively consists of only 1 component. The blue line, drawn in the middle at $\theta = \pi/2$, indicates a point where the model should realistically prefer the "EVE 2" model instead, as the generative distributions are stretched along both axes. This should in theory cause BIC-based model selection to identify equal orientations, but instead varying component shapes.

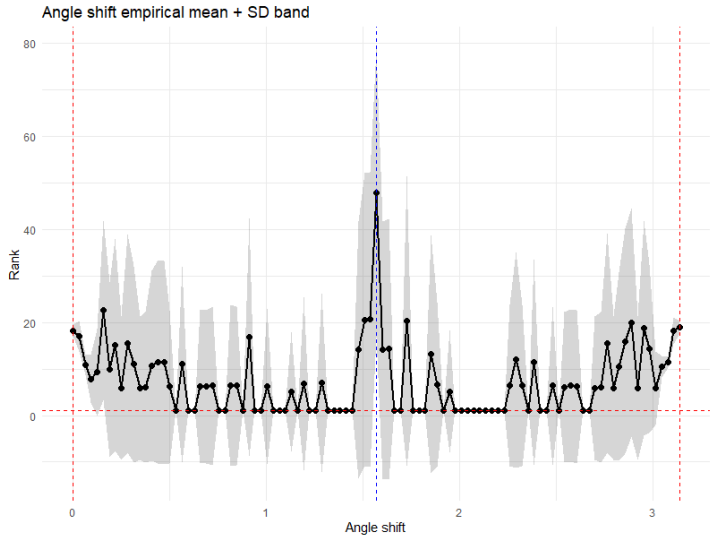

Figure 3.7: Plot of empirical mean with SD bands for ranking of the EEV 2 model. For each orientation shift value, the empirical mean and SD bands are calculated over 10 simulations.

Even though we find that the empirical mean ranks are generally worse than for the 1D-mean shift experiment, we do find that the mean rank of the true model type peaks around the points mentioned and indicated by the vertical lines in Figure 3.7. In order to look at the preferred models, we create another 100%-stacked bar chart in Figure 3.8.

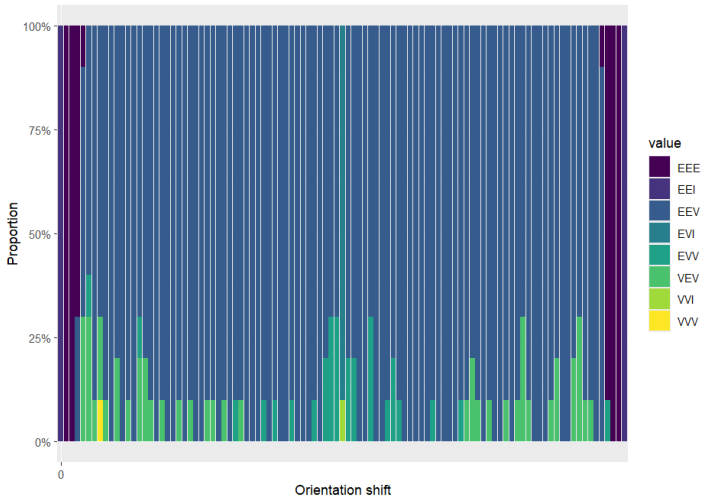

Figure 3.8: 100%-stacked bar chart of preferred model for the orientation shift experiment for $\theta \in [0, \pi]$. For each orientation shift value, 10 simulations are performed.

We find that, although Figure 3.7 seemed to show poor estimations in the desired model ranking, said model type is actually preferred quite often. That being said, the aforementioned areas around $\theta = 0, \theta = \pi/2$ and $\theta = \pi$ do cause preference for different model types. Figure 3.9 shows an example of data (and overlapping generative distributions) for which a single-component model was fitted incorrectly.

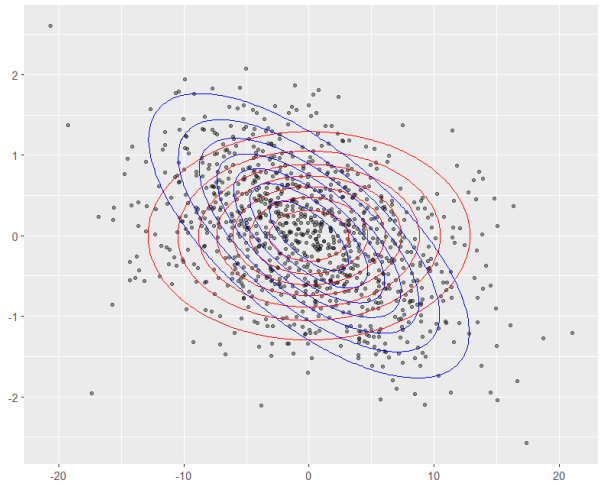

Figure 3.9: Data and generative distributions for $\theta = \frac{3\pi}{100}$. For this data, `mclust` incorrectly estimated the data to have come from a 1-component model

Once more, let us consider a 100%-stacked bar chart but for the number of components in Figure 3.10.

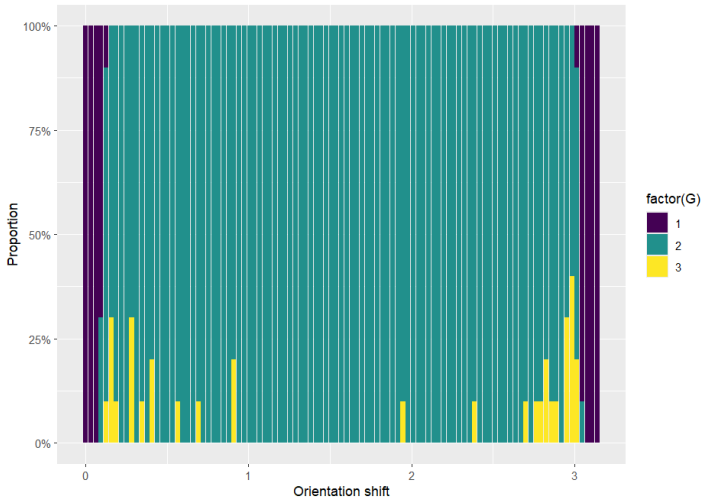

Figure 3.10: 100%-stacked bar chart of number of components for the orientation shift experiment for $\theta \in [0,\pi]$. For each orientation shift value, 10 simulations are performed.

As Figure 3.8 seemed to suggest, using BIC we do seem to be able to correctly identify the presence of 2 components quite consistently, with the exception of values of θ close to the points $\theta = 0$ and $\theta = \pi$, where again the generative distributions overlap.

3.3. Volume shifting

In this experiment, we will consider a data set containing data from 2 components following a bivariate Gaussian distribution, this time with the exact same mean vector $\boldsymbol{\mu}$. What will differ now is the volume-related constant λ_k from the decomposition in Equation 2.3. We still aspire to find 2 components, and more specifically the model type "VEE 2". The specific parameter values used for this experiment (and other relevant details) are as follows:

- **Mean vectors** $\boldsymbol{\mu}_1 = \boldsymbol{\mu}_2 = (0,0)$.
- **Covariance matrices** $\Sigma_k = \lambda_k D_k A_k D_k^T$, where:
  - $\lambda_1 = 5$ and $\lambda_2 = \lambda \in [0.1, 22.5]$ variable.
  - $A_1 = A_2 = \begin{pmatrix} 10 & 0 \\ 0 & 0.1 \end{pmatrix}$
  - $D_1 = D_2 = \begin{pmatrix} cos(\pi/6) & sin(\pi/6) \\ -sin(\pi/6) & cos(\pi/6) \end{pmatrix}$.
- **Observations per component** $n_1 = n_2 = 500$.
- **Increment**: The value of the λ-parameter increments by 0.1, i.e. $\lambda_1 = 0.1, \lambda_2 = 0.2, \lambda_3 = 0.3, ..., \lambda_{225} = 22.5$.
- **Simulations**: Per λ-value, the experiment will be run 10 times (for creation of empirical result plots).

An example of generative distributions for this volume shift experiment can be seen in the following Figure 3.11.

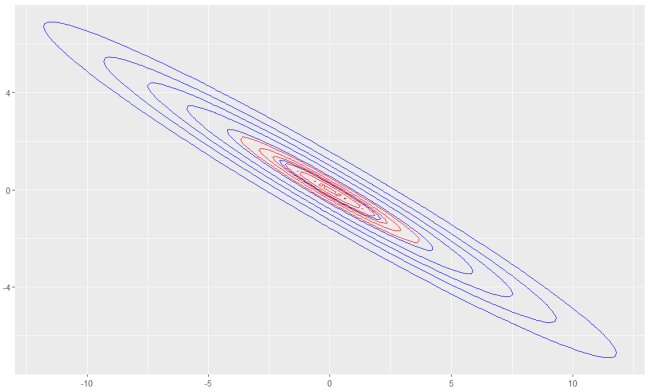

Figure 3.11: The generative distributions for orientation shifting for $\lambda = \lambda_5 = 0.5$.

Similarly to Section 3.1, we visualize the results of the described simulations in Figure 3.12 by plotting the empirical mean of the "VEE 2" model ranking and standard deviation bands. Here, a horizontal line has again been added at desired model ranking 1. Additionally, a vertical line has been added at $\lambda = 5$, as there the two components have the same parameter values and the data is thus effectively made from only one component.

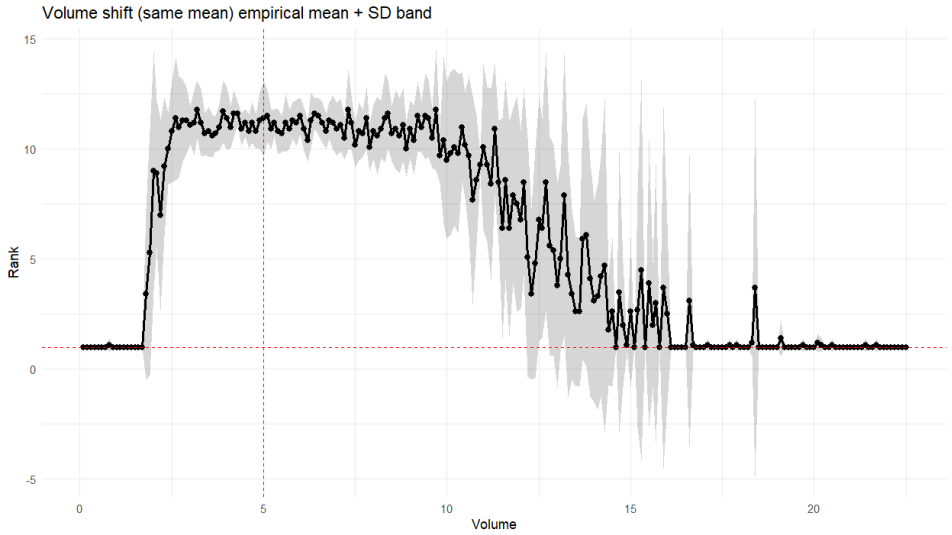

Figure 3.12: Plot of empirical mean with SD bands for ranking VEE 2 model. For each value of size parameter λ, the empirical mean and SD bands are calculated for 10 simulations.

We seem to find that the desired model type is not preferred at all when using BIC whenever the volumes of the different components are even remotely close together. That being said, it should be noted that the preferred model is identified correctly approximately where $\lambda \leq 1\frac{1}{2}$ and $\lambda \geq 16\frac{2}{3}$. Comparing these values to the constant volume value of 5, we find that $1\frac{1}{2}$ and $16\frac{2}{3}$ are smaller and bigger than said constant 5 by a factor of $3\frac{1}{3}$ respectively. Let us create another bar chart for the preferred model type to find out whether another particular model type is consistently preferred instead.

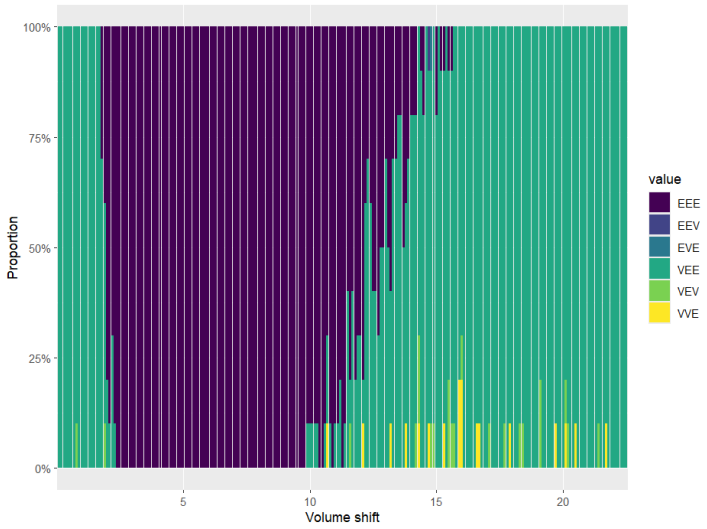

Figure 3.13: 100% -stacked bar chart of preferred model for the volume shift experiment for $\lambda \in [0.1, 22.5]$. For each volume shift value, 10 simulations are performed.

We find that, in the problem area mentioned, a single component model seems to be preferred most of the time in Figure 3.13. Although it was to be expected that the area around $\lambda = 5$ would be more likely to lead to single component model preference, the size of this region is quite large.

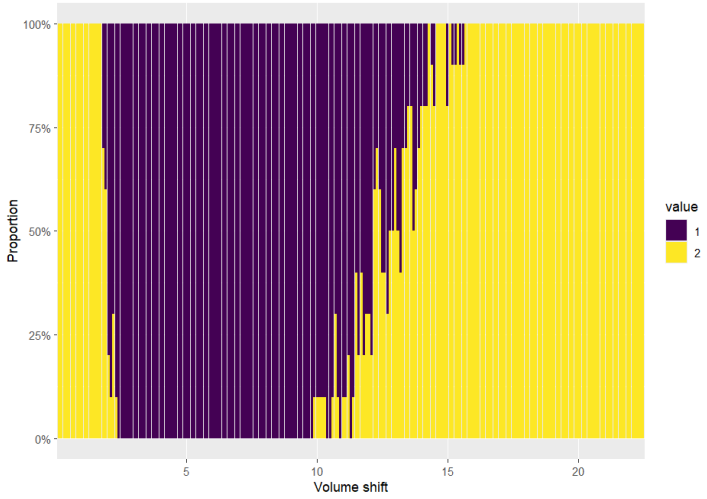

Figure 3.14: 100%-stacked bar chart of number of components for the volume shift experiment for $\lambda \in [0.1, 22.5]$. For each volume shift value, 10 simulations are performed.

In accordance to Figure 3.13 seemed to suggest, using BIC we seem to correctly identify 2 components everywhere where single component models are not at the top of the rankings. This result does make sense, as one of the two generative distributions, and in turn the generated data, is always completely overlapped by the other (ignoring possible tremendous outliers). We find that 1-component models are (incorrectly) preferred approximately in the interval $\lambda \in [2, 15]$, with said models being preferred in each of the 10 simulations approximately where $\lambda \in [2.5, 10]$.

4

Possible improvement of BIC-based selection

4.1. AIC

In the various experiments in Chapter 3 we had found that, among other regions, BIC-based model selection often identified an incorrect number of components for the preferred model as the two generative distributions overlapped more. Where this happened, `mclust` often instead preferred a single-component model over a model with 2 components which the generated observations originated from. One reason for this could be in the use of the BIC for model selection. As can be seen in Equation 2.6, BIC penalizes models for having a high number of parameters to be estimated. The incorrect conclusions made in the region means could have happened due to this strong penalty term. Added likelihood by a higher number of components would be overpowered by the extra parameters to be estimated.

For this reason, we look at a different quantity to base our model selection on. This criterion is called the Akaike Information Criterion (abbreviated to AIC) and is defined as in Equation 4.1.

$$\text{AIC} = -2\ln(L) + 2k \tag{4.1}$$

Similar to Equation 2.6, the variables indicate:

- L is the (maximized) likelihood function for the considered model, thus representing how well the model fits the data.
- k is the number of parameters to be estimated for the considered model, thus representing the complexity of using said model.

Comparing the two formulas, notice how they both contain the $-2\ln(L)$ term. They thus only differ in the penalty term, where AIC uses $2k$ and BIC $k\ln(n)$. As $\ln(n) > 2 \Leftrightarrow n > e^2 = 7.39$ and thus for any realistic data set, we see that AIC penalises complex models with larger numbers of parameters to be estimated significantly less. Based on this observation, AIC could have potential for fixing the issue mentioned. On the implementation of the AIC, it should be noted that `mclust` does not currently have built-in functionality for fitting models using AIC. For this reason, the log-likelihoods will be determined from the BIC values (which can be produced for all model types within `mclust` functionality), following Equation 4.2.

$$\text{BIC} = -2\ln(L) + k\ln(n) \Leftrightarrow \ln(L) = \frac{k\ln(n) - \text{BIC}}{2} \tag{4.2}$$

We will look to apply AIC to the 1D-mean and volume shift experiments of Chapter 3. Given the experiment setup in Section 3.2, BIC-based selection estimated the correct number of components for most parameter values. As underestimation only occurred very close to orientation parameter values $\theta = 0$ and $\theta = \pi$, there is not a significant number of underestimation for AIC-based estimation to improve upon. Furthermore, it was already found in Figure 3.10 that BIC-based estimation tended to overestimate the number of components at times. Due to the leniency of AIC as a selection criterion towards more complex models, it would be very unlikely for AIC-based estimation to improve in these areas as well.

For the 1D-mean shift experiment, intuitively it would be ideal to see that for the region where $|x| < 3$, model selection using AIC is more consistently preferring a model with 2 components. As for $x = 0$, the generative distributions have the exact same values for all parameters, it is unrealistic to hope for this region to disappear entirely. Regardless, perhaps using AIC can be more accurate over this region where BIC was the most inaccurate. For the volume shift experiment, it would be ideal to see an improvement in estimation using AIC near $\lambda = 5$, where again the generative distributions have the exact same values for all parameters. The R code for any figures created in this chapter can be found in Appendix D.

4.1.1. Using AIC for the 1D-mean shift experiment

We will once again find the preferred model using the same setup of the 1D-mean shift experiment introduced in Section 3.1, now using AIC for model selection. As a refresher we use the following parameter values:

- **Mean vectors** $\boldsymbol{\mu}_1 = (0,0)$ and $\boldsymbol{\mu}_2 = (x,0)$ with x variable with $x \in [-6,6]$.
- **Covariance matrices** $\Sigma_1 = \Sigma_2 = \begin{pmatrix} 1 & 0.5 \\ 0.5 & 1 \end{pmatrix}$.
- **Observations per component** $n_1 = n_2 = 500$.
- **Increment**: The value of the x-parameter increments by 0.1, i.e. $x_1 = -6, x_2 = -5.9, x_3 = -5.8, ..., x_{121} = 6$.
- **Simulations**: Per x-value, the experiment will be run 10 times (for creation of empirical result plots).

Running the simulations for the 121 different values of x as described, we find results as shown in Figure 4.1. Here, the empirical mean of the "EEE 2" model ranking is once again shown, in addition to empirical standard deviation bands. Furthermore, a horizontal line is drawn where $y = 1$, indicating the desired rank of the model. Another line, now vertical, is also drawn at $x = 0$, as here both components have the exact same distribution parameters and effectively create a collection of data originating from 1 component.

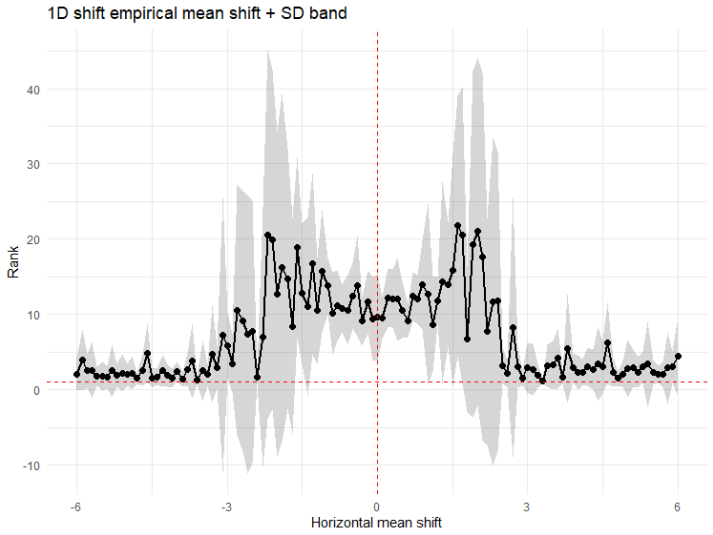

Figure 4.1: Plot of empirical mean with SD bands for ranking EEE 2 model using AIC. For each horizontal mean shift value, the empirical mean and SD bands are calculated over 10 simulations.

Upon visual inspection, one can find slightly similar behaviour to the similar plot using BIC in Figure 3.2, albeit with less certainty. Let us consider the 100%-stacked bar chart of the favoured model in Figure 4.2.

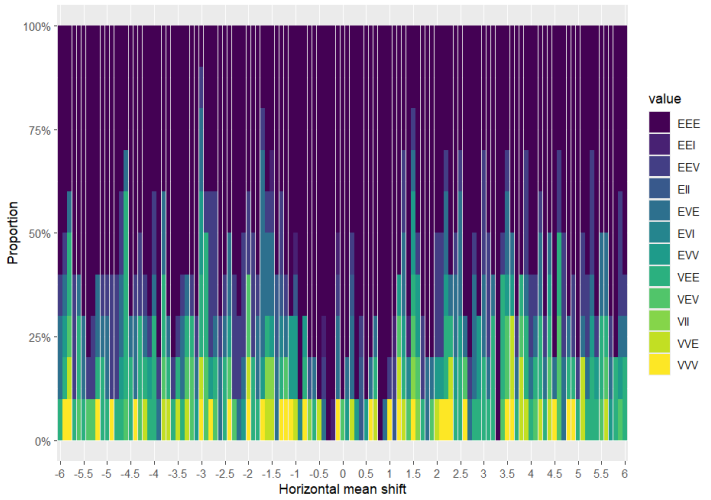

Figure 4.2: 100%-stacked bar chart of preferred model for the 1D-mean shift experiment for $x \in [-6,6]$. For each horizontal mean shift value, the empirical mean and SD bands are calculated over 10 simulations.

Where the bar chart for preferred model using BIC in Figure 3.3 indicated that the "EEE" model type was almost always preferred, Figure 4.2 seems to be very unsure. Let us finally look at the 100%-stacked bar chart for number of components in Figure 4.3.

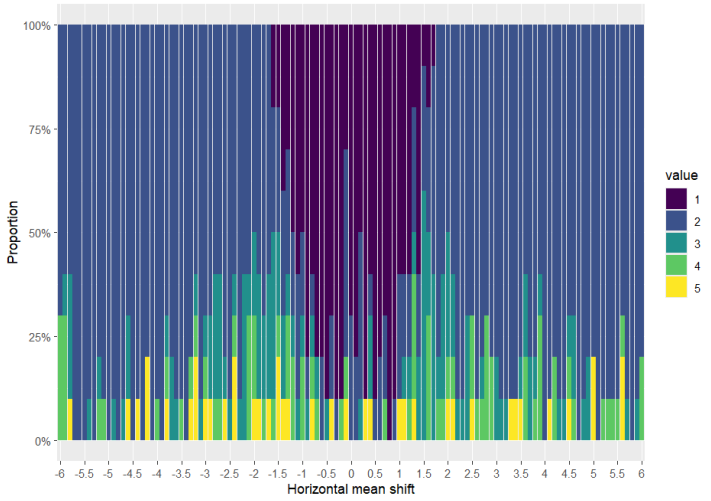

Figure 4.3: 100%-stacked bar chart of number of components for the 1D-mean shift experiment for $x \in [-6,6]$. For each horizontal mean shift value, the empirical mean and SD bands are calculated over 10 simulations.

Contrary to the component bar chart when using BIC in Figure 3.4, we find that even in the region $|x| \leq 1.5$ mentioned earlier where using BIC almost never preferred a 2-component model, using AIC does often still identify a model with 2 components to be optimal for the data set. That being said, a drawback of AIC does become apparent, as models with more than 2 components are often incorrectly identified as optimal. This behaviour falls in line with the observation made that AIC penalizes models with higher components significantly less than BIC-based model selection. Intuitively, it seems like BIC-based model selection tends to underestimate the optimal number of components, whereas AIC-based model selection tends to overestimate said quantity. In the case where one of these downsides is significantly less of a problem (for example if the computation time for model fitting is virtually irrelevant), such context could lead to a preference for using one of these criteria over the other.

In order to compare AIC- and BIC-based number of component selection, let us plot the following quantity for each horizontal mean shift value, which we will call Mean Component Accuracy Difference (MCAD), to use as a metric:

$$\text{MCAD} = |C_{\text{BIC}} - C| - |C_{\text{AIC}} - C|. \tag{4.3}$$

Here:

- C_{AIC} represents the mean number of components of the AIC-selection favoured model.
- C_{BIC} represents the mean number of components of the favoured model through BIC selection.
- C represents the true number of components of which the observations are present in the data provided.

Note that as given the setup of this experiment, the value of C is constant at 2. The MCAD is a singular value that compares how close the number of components estimated is on average between AIC- and BIC-based selection. As a higher value of $|C_{\text{AIC}} - C|$ means that AIC is on average inaccurate in its estimations (and similar for $|C_{\text{BIC}} - C|$), positive MCAD shows closer estimations by AIC on average and vice versa. Figure 4.4 plots the MCAD over the same horizontal mean shift closed interval $[-6,6]$. Given the definition of MCAD, a horizontal line has been added at MCAD = 0 to indicate equal average accuracy.

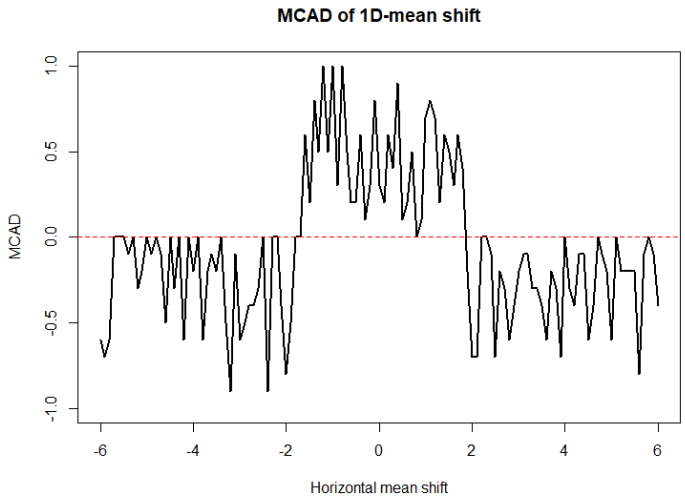

Figure 4.4: MCAD plotted for $x \in [-6,6]$. For each horizontal mean shift value, the MCAD is calculated using both 10 simulations using AIC and 10 using BIC.

Although BIC-based number of component selection consistently outperforms roughly for $|x| \geq 2$, AIC-based selection has the edge roughly where $|x| < 2$. Comparing this to our results in Figure3.4, we find that BIC-based selection estimated incorrect models with 1 component in almost exactly this interval where AIC-based selection outperforms. It thus seems like, in this 1D-mean shift setting, using AIC-based selection can indeed be useful in areas where BIC-based selection tends to struggle.

This positive result for AIC-based selection is despite the fact that, for the area where BIC-based selection struggles, it only seems to consider 1- and 2-component models. This in turn means that $|C_{\text{BIC}} - C| \leq 1$, whereas the similar quantity $|C_{\text{AIC}} - C|$ can be much larger for AIC-based clustering. The reason behind the latter quantity being lower where $|x| < 2$ seems to be due to AIC both under- and overestimating at times, which in a metric like MCAD cancel each other out in an advantageous fashion for AIC-based model selection.

4.1.2. Using AIC for the volume shift experiment

The same experiment setup will be used as in the volume shift experiment using BIC in Section 3.3, thus with the following parameter values:

- **Mean vectors** $\boldsymbol{\mu}_1 = \boldsymbol{\mu}_2 = (0,0)$.
- **Covariance matrices** $\Sigma_k = \lambda_k D_k A_k D_k^T$, where:
  - $\lambda_1 = 5$ and $\lambda_2 = \lambda \in [0.1, 22.5]$ variable.
  - $A_1 = A_2 = \begin{pmatrix} 10 & 0 \\ 0 & 0.1 \end{pmatrix}$

– $D_1 = D_2 = \begin{pmatrix} cos(\pi/6) & sin(\pi/6) \\ -sin(\pi/6) & cos(\pi/6) \end{pmatrix}$.

- **Observations per component** $n_1 = n_2 = 500$.
- **Increment**: The value of the λ-parameter increments by 0.1, i.e. $\lambda_1 = 0.1, \lambda_2 = 0.2, \lambda_3 = 0.3, ..., \lambda_{225} = 22.5$.
- **Simulations**: Per λ-value, the experiment will be run 10 times (for creation of empirical result plots).

For the 225 different values of λ, Figure 4.5 shows the empirical mean of the "VEE 2" model ranking with empirical standard deviation bands. Again, a horizontal and vertical line have been added at the best model ranking possible and the parameter value λ of the unchanging component, respectively.

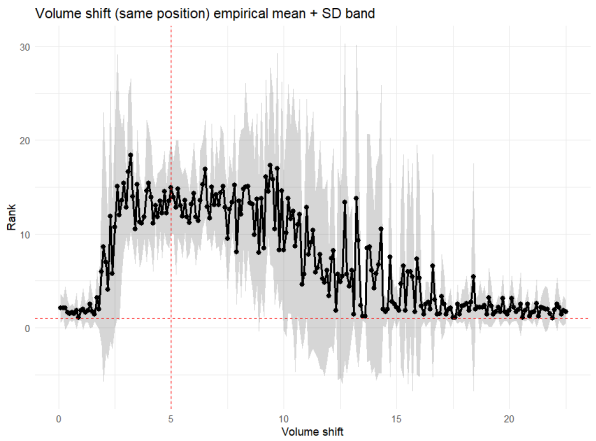

Figure 4.5: Plot of empirical mean with SD bands for ranking VEE 2 model using AIC. For each volume shift value, the empirical mean and SD bands are calculated over 10 simulations.

Visually we seem to find a slightly similar development of the empirical mean over the different values of λ, although less stable than at the ends where the target "VEE 2" model type was almost always preferred when using BIC in Figure 3.12. Let us now look at 100%-stacked bar charts for both the estimated model and number of components using AIC in Figure 4.6 and Figure 4.7 respectively.

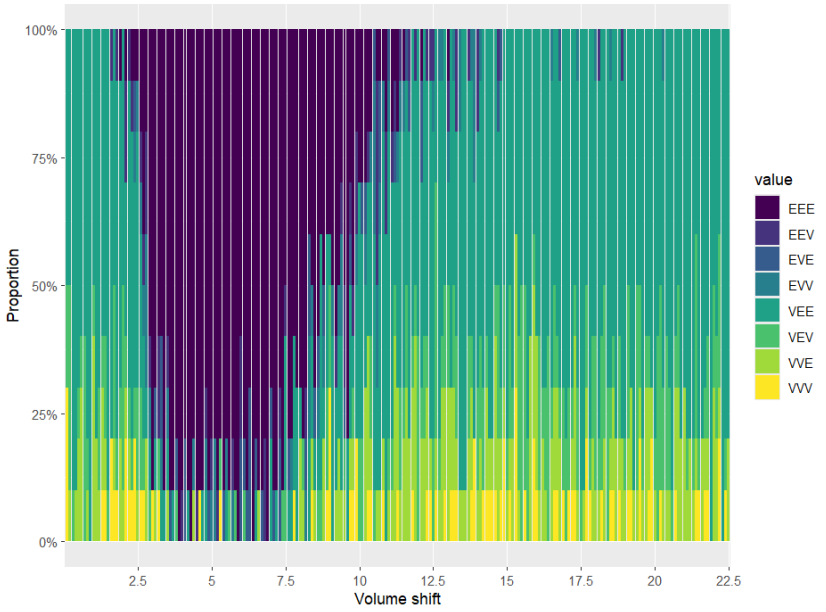

Figure 4.6: 100%-stacked bar chart of preferred model type for the volume shift experiment using AIC for $\lambda \in [0.1, 22.5]$. For each volume shift value, 10 simulations are performed.

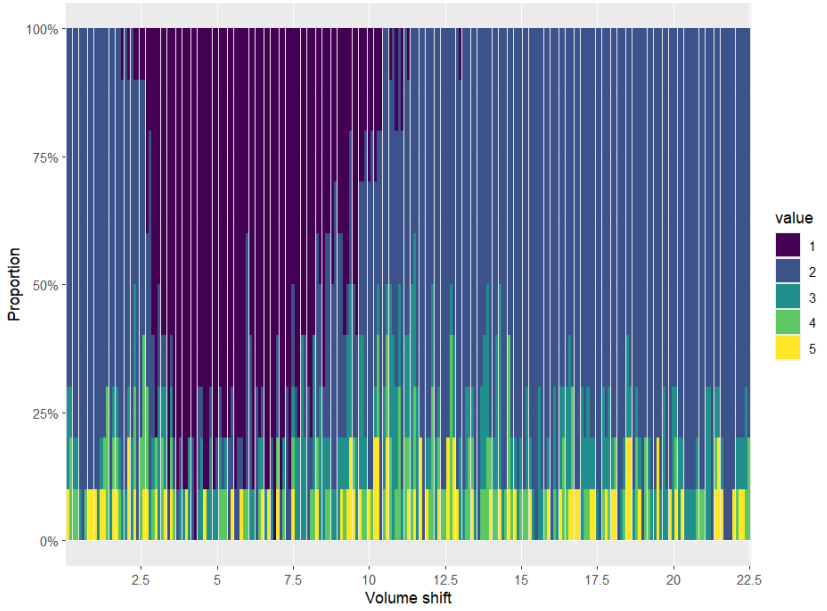

Figure 4.7: 100%-stacked bar chart of number of components for the volume shift experiment using AIC for $\lambda \in [0.1, 22.5]$. For each volume shift value, 10 simulations are performed.

As seen before in Figure 4.3 when applying AIC in the 1D-mean shift experiment in Section 4.1.1, it tends to overestimate the true number of components, even preferring 5-component models throughout all of the possible values of λ. Let us plot the MCAD again as defined in Equation 4.3 for this volume shift experiment, as can be seen in Figure 4.8.

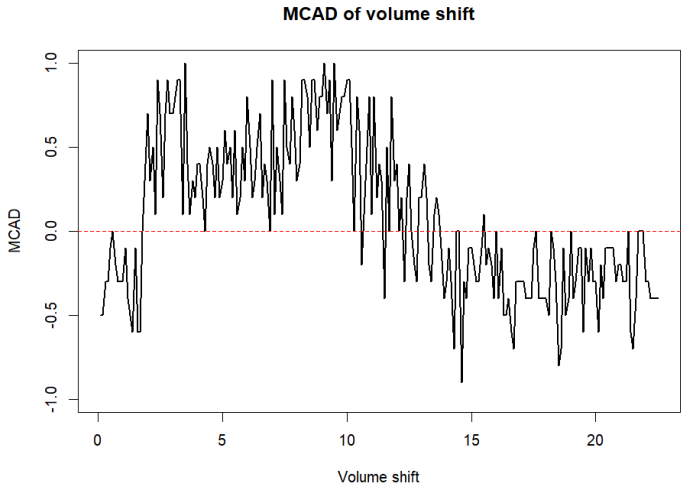

Figure 4.8: MCAD plotted for $\lambda \in [0.1, 22.5]$. For each volume shift value, the MCAD is calculated using both 10 simulations using AIC and 10 using BIC.

Remember how, in the equivalent MCAD plot for BIC-based estimations of Figure 3.14, it was found that preferences for 1-component model were found by `mclust` approximately where $\lambda \in [2, 15]$. Having used AIC now, a better average performance of the number of components estimated is found over almost all of this interval. Outside of this interval, the BIC-based selection performs better once again. Regardless, it again seems like AIC-based selection can provide a significant benefit there where BIC-based selection struggles.

4.2. Bivariate copula mixture models

In this section, we will test `mclust` further in its robustness through misspecification scenarios, where we will simulate data sets using bivariate copula mixture models. Although mixture models are nothing new at this point, copulas require some introduction.

So far, we have seen and used bivariate Gaussian distributions in the component specification in all of our experiments. One of the characteristics of such bivariate Gaussian distributions is that for resulting random variable pairs (X, Y), their dependence always follows an ellipse-like correlation. Even with the decomposition that we had discussed in Equation 2.3, this correlation is always found regardless of what values we give for the volume-, orientation- and shape-related components. Such guaranteed ellipsoidal shape of data generated from a bivariate Gaussian distribution is what was kept in mind with the existing selection criteria BIC and AIC. With copulas however, we will change this shape to see how `mclust` performs.

For the purposes of this research, advanced knowledge of copulas is not required seeing as it is mainly the bivariate distribution function that we sample from to generate data. Regardless, an observation using a (bivariate) copula can be generated as follows:

1. **Choose a copula**: There are plenty of different types of copulas, each showing a different kind of dependence structure between variables (e.g. a symmetric dependence or a dependence where one tail has stronger dependence than the other). This dependence can often be tuned further (e.g. making the dependence stronger or weaker) by changing the value of the parameter of the chosen copula.

2. **Sample from copula**: Following the dependence structure of the chosen copula, a pair of values (u, v) is generated, with $0 \leq u \leq 1$ and $0 \leq v \leq 1$. Although individually, they are uniformly distributed over this $[0,1]$ interval, the dependence structure of the chosen copula is reflected in the pairing. For example, for a copula with strong dependence on the lower tail, low values for u tend to imply a low value of v, and vice versa. As one may expect, this is also the case for v.

3. **Couple with marginal distributions**: An observation (X, Y) is finally created by choosing 2 marginal distributions, which for this research will both be univariate Gaussian but may be different (even different from each other). Then given the univariate distributions chosen for X and Y, with (cumulative) distribution functions $F_X(x)$ and $F_Y(y)$, the paired observation then becomes $(F_X^{-1}(u), F_Y^{-1}(v))$.

As we can see, copulas never directly interact with the marginal distributions, instead providing values from which the marginal distributions (more specifically the inverse of their cumulative distribution functions) generate observations. Only once these paired observations are looked at as a whole does the impact of the copula become clear. In Figure 4.9, the shapes and thus dependence structures of various different types of copulas can be seen.

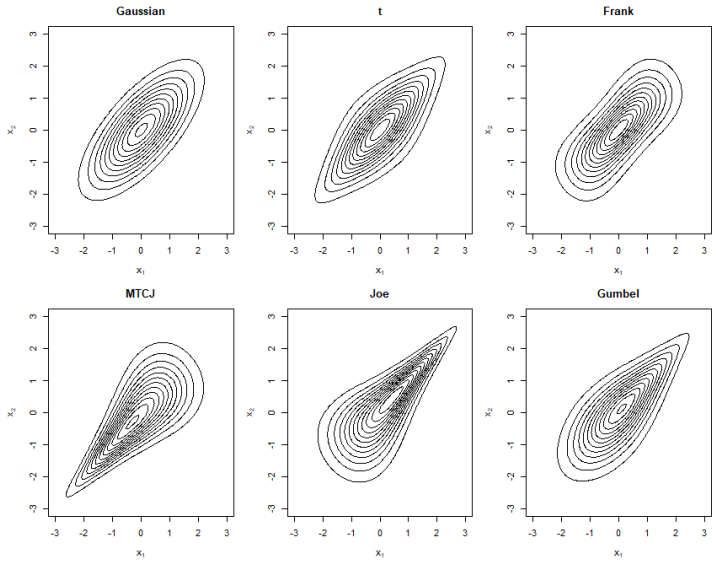

Figure 4.9: Contour plots of Gaussian, t, Frank, MTCJ, Joe and Gumbel copula. These copulas have copula parameter value(s) 0.71, (4, 0.71), 5.74, 2, 2.86 and 2 respectively.

For more information about copulas, consult Nelsen (2006).

4.2.1. 1D-mean shift experiment with bivariate copula mixture models

We will use bivariate copula mixture models mainly in a setting similar to the 1D-mean shift experiment we did in Section 3.1. More specifically, we will use 2 components, one of which using the Mardia-Takahasi-Clayton-Cook-Johnson (MTCJ) copula and the other with the Joe copula. Furthermore, the details are as follows:

- **Mean vectors** $\boldsymbol{\mu}_1 = (0,0)$ and $\boldsymbol{\mu}_2 = (x,0)$ with x variable with $x \in [-6,6]$.
- **Variances** $\sigma_1 = \sigma_2 = 1$. Remember how bivariate copula mixture models cannot have a covariance matrix defined as the marginal distributions are not dependent in the step of the process where they are generated.
- **Observations per component** $n_1 = n_2 = 500$.
- **Increment**: The value of the x-parameter increments by 0.1, i.e. $x_1 = -6, x_2 = -5.9, x_3 = -5.8, ..., x_{121} = 6$.
- **Simulations**: Per x-value, the experiment will be run 10 times (for creation of empirical result plots).
- **Copula parameters**: For both copulas and in turn both components, a copula parameter value of 3 is used.

In Figure 4.10, the 100%-stacked bar chart can be found for this experiment.

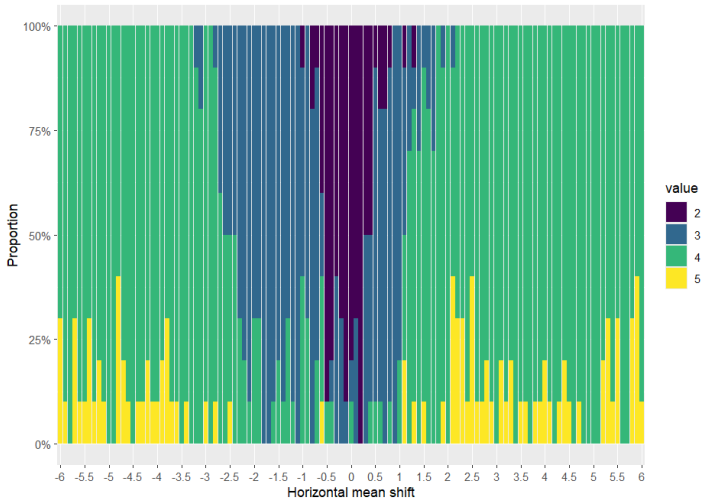

Figure 4.10: 100%-stacked bar chart of number of components of BIC for $x \in [-6,6]$. For each horizontal mean shift value, 10 simulations are performed.

We find that 2-component models are very rarely estimated using `mclust`, instead often preferring models with more components. In fact, even when the horizontal mean shift is 0, 1-component models are never estimated. It would not be too surprising to think that this overestimation of the number of components comes from the non-elliptical shape of the component data, as the models considered by `mclust` are not representative of the copula-based data. An example of such copula-based data can be seen in Figure 4.11.

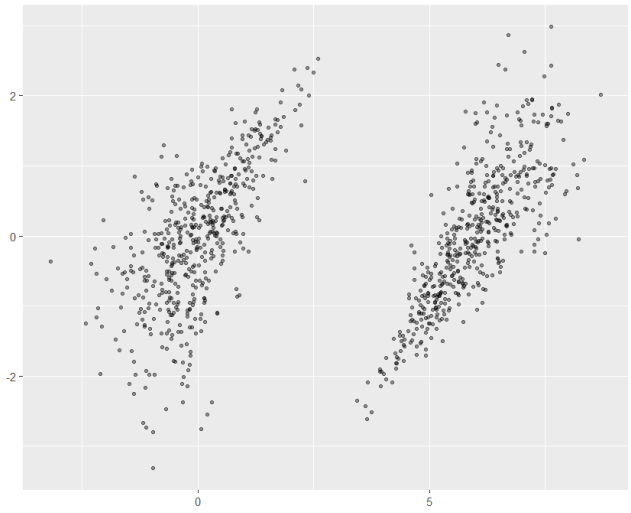

Figure 4.11: Data and generative distributions for $x = 6$. For this data, `mclust` incorrectly estimated the data to have come from a 4-component model. Using AIC, a 5-component model is preferred.

In Figure 4.12, the overestimation of the number of components described are again found when using the AIC criterion.

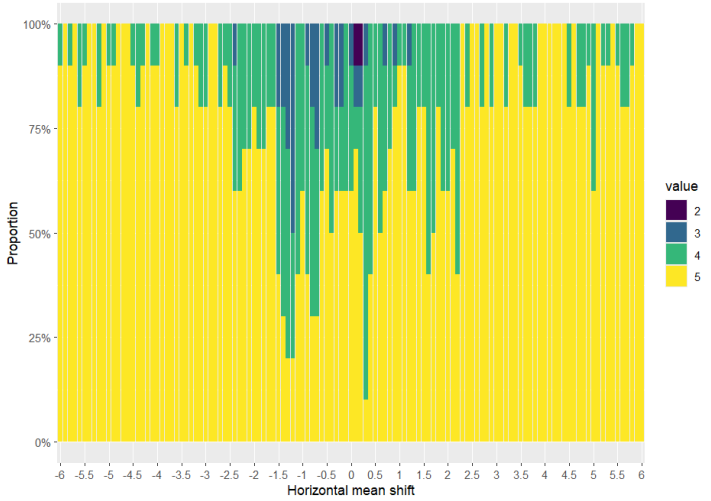

Figure 4.12: 100%-stacked bar chart of number of components of AIC for $x \in [-6.6]$. For each horizontal mean shift value, 10 simulations are performed.

Seemingly, attempting to fit to the non-elliptical shape of the copula-based data, AIC-based model selection prefers models with more components than BIC-based selection at almost all horizontal mean shift levels. This is not surprising given what we know about AIC, but it is still significantly worse than BIC-based model selection. It is important to remember that only 1-, 2-, 3-, 4- and 5-component models were considered for the sake of computational time. That being said, the performance found is tremendously worse than for BIC-based model selection and it is safe to assume that considering models with more components would not lead to different results.

5

Conclusion

In this thesis the robustness of BIC-based number of component selection was tested for Gaussian mixture models. In Chapter 3, we found that this selection criterion tends to struggle whenever different components have similar parameter values. For each experiment, intervals were identified in which this problem occurred. For the 1D-mean shift and volume shift experiments of Section 3.1 and Section 3.3, it was found in Section 4.1.1 and Section 4.1.2 that AIC-based selection tended to perform better on average than BIC-based selection in the regions where BIC often estimated the wrong number of components. This improvement came at the cost of worse average accuracy in regions where BIC-based selection had near-perfect performance, as AIC-based selection tends to overestimate the number of components. Finally in Section 4.2.1, copulas were used to generate data with non-elliptical shapes. In that setting, the 1D mean-shift experiment generally did not yield accurate results for either BIC or AIC, with AIC performing even worse than BIC.

For future work, one could consider combining BIC- and AIC-based estimation, for example by using the estimated parameters of the best fitting model across different model types. As we had found that AIC generally outperforms BIC-based selection when the parameter values of components are similar, this similarity could potentially be expressed in some quantity based on which the selection criterion is chosen. Furthermore, the experiments performed in this thesis could be replicated with more than 2 components, although more decisions should be made about the ways in which these components differ among multiple iterations for meaningful results. Finally, experiments could be performed where components have unequal component probabilities. For simplicity, it was always assumed for these probabilities to be equal, but varying them might lead to interesting results.

A

Used packages

```
#install.packages("mclust")
library(mclust)

#install.packages("MASS")
library(MASS)

#install.packages("gclus")
library(gclus)

#install.packages("rcompanion")
library(rcompanion)

#install.packages("ggplot2")
library(ggplot2)

#install.packages("mvtnorm")
library(mvtnorm)

#install.packages("copula")
library(copula)

#install.packages("future")
#install.packages("future.apply")
#install.packages("furrr")
library(future)
library(future.apply)
library(furrr)

#install.packages("viridis")
library(viridis)

#install.packages(c("dplyr", "tidyr", "purrr"))
library(dplyr)
library(tidyr)
library(purrr)

#remotes::install_github("oezgesahin/vineclust")
library(vineclust)
```

B

Code used in Chapter 2

B.1. Examples for motivating mixture models

```
## Creates Figure 2.1
n = 1000
proportion1 = 0.3
proportion2 = 0.5

set.seed(12345)
x = rnorm(n*proportion1, mean = 7, sd = 1)
y = rnorm(n*proportion2, mean = 17, sd = 0.5)
z = rnorm(n*(1-proportion1-proportion2), mean = 29, sd = 2)
data = c(x,y,z)

hist(data, main = "",
      xlab = "",
      ylim = c(0,0.5),
      ylab = "",
      probability = TRUE,
      breaks = seq(0,35, by = 0.5))

## Creates Figure 2.2
n = 1000
proportion1 = 0.3
proportion2 = 0.5

set.seed(12345)
x = rnorm(n*proportion1, mean = 7, sd = 1)
y = rnorm(n*proportion2, mean = 17, sd = 0.5)
z = rnorm(n*(1-proportion1-proportion2), mean = 29, sd = 2)
data = c(x,y,z)

hist(data, main = "",
      xlab = "",
      ylim = c(0,0.5),
      ylab = "",
      probability = TRUE,
      breaks = seq(0,35, by = 0.5))
x = seq(0, max(data), length.out = 1000)
points = 2000
#G = 1 curve
```

```
model1 = Mclust(data, G = 1)
mean1 = model1$parameters$mean
var1 = model1$parameters$variance$sigmasq
curve(dnorm(x, mean = mean1, sd = sqrt(var1)), add = TRUE, col = "red",  lwd = 2, n = points)

#G = 3 Curve
model3 = Mclust(data, G = 3)
pro3 = model3$parameters$pro
mean3 = model3$parameters$mean
var3 = model3$parameters$variance$sigmasq

curve(pro3[1] * dnorm(x, mean3[1], sqrt(var1[1])) +
      pro3[2] * dnorm(x, mean3[2], sqrt(var1[2])) +
      pro3[3] * dnorm(x, mean3[3], sqrt(var1[3]))
      , add = TRUE, col = "blue",  lwd = 2, n = points)

## Creates Figure 2.3
muDogs = c(0.55, 25)
SigmaDogs = matrix(c(0.0625, 1.5,
                     1.5, 100), nrow = 2)
set.seed(12345)
clusDogs = mvrnorm(n = 100, mu = muDogs, Sigma = SigmaDogs)

muEle = c(3, 5000)
SigmaEle = matrix(c(0.16, 306,
                    306, 810000), nrow = 2)
set.seed(12345)
clusEle = mvrnorm(n = 100, mu = muEle, Sigma = SigmaEle)

muGir = c(5.5, 800)
SigmaGir = matrix(c(0.36, 63,
                    63, 22500), nrow = 2)
set.seed(12345)
clusGir = mvrnorm(n = 100, mu = muGir, Sigma = SigmaGir)



data = rbind(clusDogs, clusEle, clusGir)
plot(data[, 2], data[, 1],
     xlab = "Weight␣in␣kilograms",
     ylab = "Height␣in␣meters")

## Creates Figure 2.4
muDogs = c(0.55, 25)
SigmaDogs = matrix(c(0.0625, 1.5,
                     1.5, 100), nrow = 2)
set.seed(12345)
clusDogs = mvrnorm(n = 100, mu = muDogs, Sigma = SigmaDogs)

muEle = c(3, 5000)
SigmaEle = matrix(c(0.16, 306,
                    306, 810000), nrow = 2)
clusEle = mvrnorm(n = 100, mu = muEle, Sigma = SigmaEle)

muGir = c(5.5, 800)
SigmaGir = matrix(c(0.36, 63,
                    63, 22500), nrow = 2)
```

```
clusGir = mvrnorm(n = 100, mu = muGir, Sigma = SigmaGir)



data = rbind(clusDogs, clusEle, clusGir)
plot(data[, 2], data[, 1],
     xlab = "Weight in kilograms",
     ylab = "Height in meters")

data_temp = data
data = scale(data)

model1G = Mclust(data, G = 1)
model3G = Mclust(data, G = 3)

heights = data[,1]
weights = data[,2]
#G = 1 curve
xg <- seq(min(weights), max(weights) *1.1, length.out = 200)
yg <- seq(min(heights), max(heights) *1.1, length.out = 200)

grid <- expand.grid(
  weight = xg,
  height = yg
)

# Parameters from fitted 1-component model
mu <- model1G$parameters$mean
Sigma <- model1G$parameters$variance$sigma[,,1] #Splice to remove last dimension added by mclust

# Evaluate density
grid$z <- dmvnorm(
  x = as.matrix(grid),
  mean = mu,
  sigma = Sigma
)

#Work to plot in original units
center <- attr(data, "scaled:center")
scalev <- attr(data, "scaled:scale")

grid_orig <- grid

grid_orig$weight <- grid$weight * scalev[2] + center[2]
grid_orig$height <- grid$height * scalev[1] + center[1]

# Plot
ggplot(data.frame(
  weight = data_temp[,2],
  height = data_temp[,1]
), aes(weight, height)) +
  geom_point(alpha = 0.5) +
  geom_contour(
    data = grid_orig,
    aes(x = weight, y = height, z = z),
    color = "red"
```

```
  ) +
  labs(
    x = "Weight in kilograms",
    y = "Height in meters"
  )

## Creates Figure 2.5
muDogs = c(0.55, 25)
SigmaDogs = matrix(c(0.0625, 1.5,
                     1.5, 100), nrow = 2)
set.seed(12345)
clusDogs = mvrnorm(n = 100, mu = muDogs, Sigma = SigmaDogs)

muEle = c(3, 5000)
SigmaEle = matrix(c(0.16, 306,
                    306, 810000), nrow = 2)
clusEle = mvrnorm(n = 100, mu = muEle, Sigma = SigmaEle)

muGir = c(5.5, 800)
SigmaGir = matrix(c(0.36, 63,
                    63, 22500), nrow = 2)
clusGir = mvrnorm(n = 100, mu = muGir, Sigma = SigmaGir)



data = rbind(clusDogs, clusEle, clusGir)
plot(data[, 2], data[, 1],
     xlab = "Weight in kilograms",
     ylab = "Height in meters")

data_temp = data
data = scale(data)

model1G = Mclust(data, G = 1)
model3G = Mclust(data, G = 3)

heights = data[,1]
weights = data[,2]

xg <- seq(min(weights), max(weights) *1.1, length.out = 200)
yg <- seq(min(heights), max(heights) *1.1, length.out = 200)

grid <- expand.grid(
  weight = xg,
  height = yg
)

#Work to plot in original units
center <- attr(data, "scaled:center")
scalev <- attr(data, "scaled:scale")

grid_orig <- grid

grid_orig$weight <- grid$weight * scalev[2] + center[2]
grid_orig$height <- grid$height * scalev[1] + center[1]
```

```
# G = 3 Curve

# Extract fitted parameters
means <- model3G$parameters$mean
sigmas <- model3G$parameters$variance$sigma

#Create separate grids
grid <- expand.grid(
  height = seq(min(data[,1]), max(data[,1]), length.out = 200),
  weight = seq(min(data[,2]), max(data[,2]), length.out = 200)
)

grid1 <- grid
grid1$z <- dmvnorm(as.matrix(grid), means[,1], sigmas[,,1])

grid2 <- grid
grid2$z <- dmvnorm(as.matrix(grid), means[,2], sigmas[,,2])

grid3 <- grid
grid3$z <- dmvnorm(as.matrix(grid), means[,3], sigmas[,,3])

ggplot(data.frame(weight = data[,2],
                  height = data[,1]),
       aes(weight, height)) +
  geom_point(alpha = 0.5) +

  geom_contour(data = grid1,
               aes(weight, height, z = z),
               color = "red") +
  geom_contour(data = grid2,
               aes(weight, height, z = z),
               color = "blue") +
  geom_contour(data = grid3,
               aes(weight, height, z = z),
               color = "darkgreen") +
  labs(
    x = "Weight␣(scaled)",
    y = "Height␣(scaled)"
  )
```

B.2. Examples of types of Gaussian mixture models

```
## Creates Figure 2.6
lambda1 = 2
lambda2 = 1
A = matrix(c(4, 0,0, 0.25), nrow = 2)

angle = pi/4
D = matrix(c(cos(angle), -sin(angle),
             sin(angle), cos(angle)), nrow = 2)

cov1 = lambda1 * D %*% A %*% t(D)
cov2 = lambda2 * D %*% A %*% t(D)

clus1 = mvrnorm(n = 150, mu = c(0,0), Sigma = cov1)
clus2 = mvrnorm(n = 150, mu = c(0,7.5), Sigma = cov2)
```

```
xlim = range(c(clus1[,1], clus2[,1]))
ylim = range(c(clus1[,2], clus2[,2]))

plot(clus1[,1], clus1[,2],
     col = "red", pch = 16,
     xlab = "", ylab = "",
     xlim = xlim, ylim = ylim)

points(clus2[,1], clus2[,2],
       col = "blue", pch = 16)

## Creates Figure 2.7
xSeq= seq(-14,14, length.out = 200)
ySeq= seq(-14,14, length.out = 200)
grid = expand.grid(xSeq = xSeq, ySeq = ySeq)
coords = cbind(grid$xSeq, grid$ySeq)

mean1 = c(-2,4)
A1 = matrix(c(10, 0,
              0, 0.1), nrow = 2)
lambda1 = 4
D1 = matrix(c(cos(pi/4), -sin(pi/4),
              sin(pi/4), cos(pi/4)), nrow = 2)

mean2 = c(2,5)
A2 = matrix(c(3, 0,
              0, 1), nrow = 2)
lambda2 = 1.5
D2 = matrix(c(cos(pi/1.5), -sin(pi/1.5),
              sin(pi/1.5), cos(pi/1.5)), nrow = 2)


mean3 = c(-3,-3)
A3 = matrix(c(4, 0,
              0, 4), nrow = 2)
lambda3 = 2.5
D3 = matrix(c(cos(pi/3), -sin(pi/3),
              sin(pi/3), cos(pi/3)), nrow = 2)

grid1 = grid
grid1$z = dmvnorm(coords, mean1, lambda1*D1%*%A1%*%t(D1))
grid2 = grid
grid2$z = dmvnorm(coords, mean2, lambda2*D2%*%A2%*%t(D2))
grid3 = grid
grid3$z = dmvnorm(coords, mean3, lambda3*D3%*%A3%*%t(D3))

ggplot() +
  geom_contour(
    data = grid1,
    aes(xSeq, ySeq, z = z),
    color = "red"
  ) +
  geom_contour(
    data = grid2,
```

```
    aes(xSeq, ySeq, z = z),
    color = "blue"
  ) +
  geom_contour(
    data = grid3,
    aes(xSeq, ySeq, z = z),
    color = "green"
  ) +
  coord_fixed() +
  coord_cartesian(xlim = c(-13, 10), ylim = c(-9, 13)) +
  labs(
    x = "",
    y = ""
  ) +
  theme(
    axis.text.x = element_blank(),
    axis.text.y = element_blank()
  )

## Creates Figure 2.8
xSeq= seq(-14,14, length.out = 200)
ySeq= seq(-14,14, length.out = 200)
grid = expand.grid(xSeq = xSeq, ySeq = ySeq)
coords = cbind(grid$xSeq, grid$ySeq)

lambda = 3
D = matrix(c(1, 0,
             0, 1), nrow = 2)


mean1 = c(-8,4)
A1 = matrix(c(3,0,
              0,1), nrow = 2)

mean2 = c(5,5)
A2 = matrix(c(2,0,
              0,6), nrow = 2)


mean3 = c(-3,-2)
A3 = matrix(c(4,0,
              0,4), nrow = 2)

grid1 = grid
grid1$z = dmvnorm(coords, mean1, lambda*D%*%A1%*%t(D))
grid2 = grid
grid2$z = dmvnorm(coords, mean2, lambda*D%*%A2%*%t(D))
grid3 = grid
grid3$z = dmvnorm(coords, mean3, lambda*D%*%A3%*%t(D))

ggplot() +
  geom_contour(
    data = grid1,
    aes(xSeq, ySeq, z = z),
    color = "red"
  ) +
```

```
  geom_contour(
    data = grid2,
    aes(xSeq, ySeq, z = z),
    color = "blue"
  ) +
  geom_contour(
    data = grid3,
    aes(xSeq, ySeq, z = z),
    color = "green"
  ) +
  coord_fixed() +
  coord_cartesian(xlim = c(-13, 10), ylim = c(-9, 13)) +
  labs(
    x = "",
    y = ""
  ) +
  theme(
    axis.text.x = element_blank(),
    axis.text.y = element_blank()
  )

## Creates Figure 2.9
xSeq= seq(-14,14, length.out = 200)
ySeq= seq(-14,14, length.out = 200)
grid = expand.grid(xSeq = xSeq, ySeq = ySeq)
coords = cbind(grid$xSeq, grid$ySeq)

D = matrix(c(1, 0,
             0, 1), nrow = 2)
A = D


mean1 = c(-5,4)
lambda1 = 12

mean2 = c(7,5)
lambda2 = 5.5

mean3 = c(-3,-10)
lambda3 = 1.25

grid1 = grid
grid1$z = dmvnorm(coords, mean1, lambda1*D%*%A%*%t(D))
grid2 = grid
grid2$z = dmvnorm(coords, mean2, lambda2*D%*%A%*%t(D))
grid3 = grid
grid3$z = dmvnorm(coords, mean3, lambda3*D%*%A%*%t(D))

ggplot() +
  geom_contour(
    data = grid1,
    aes(xSeq, ySeq, z = z),
    color = "red"
  ) +
  geom_contour(
    data = grid2,
```

```
    aes(xSeq, ySeq, z = z),
    color = "blue"
  ) +
  geom_contour(
    data = grid3,
    aes(xSeq, ySeq, z = z),
    color = "green"
  ) +
  coord_fixed() +
  coord_cartesian(xlim = c(-13, 13), ylim = c(-13, 13)) +
  labs(
    x = "",
    y = ""
  ) +
  theme(
    axis.text.x = element_blank(),
    axis.text.y = element_blank()
  )
```

C

Code used in Chapter 3

C.1. Helper functions for BIC-experiments

```
#Fits and returns the favoured model according to data and with one
#of the cluster numbers considered, specified in clusterNums
#data: Data for which model is to be chosen
#clusterNums: Number, or vector of numbers
#indicating allowed numbers of clusters to be considered
#Returns: String showing chosen model (format: "EVE 2")
FitFavModel = function(data, clusterNums) {
  model = Mclust(data, G = clusterNums, verbose = FALSE)
  #Note: Verbose = FALSE hides progress bar
  clust = model$G
  modelName = model$modelName
  return(paste(modelName, clust))
}

#Finds and returns ranking of model through given BIC values
#modelName: Name of model for which
index will be found (format: "EVE␣2")
#BICValues: Matrix of  values of different models
#Returns: (Absolute) ranking of model given
modelRankingBIC = function(modelName, BICVals) {
  modelSplit = strsplit(modelName, "␣")
  modelType = modelSplit[[1]][1]
  clusters = as.numeric(modelSplit[[1]][2])

  df = as.data.frame(as.table(BICVals))
  colnames(df) = c("G", "model", "BIC")
  df_sorted = df[order(-df$BIC), ]
  return(which(df_sorted$G == clusters & df_sorted$model == modelType))
}

#Finds and returns favoured model name given list of BIC values (format: "EVE 2")
#BICValues: Matrix of BIC values of different models
#Returns: Name of favoured model type with amount of components
favouredModelBIC = function(BICVals) {
  df = as.data.frame(as.table(BICVals))
  colnames(df) = c("G", "model", "BIC")
  df_sorted = df[order(-df$BIC), ]
```

```
  bestModel = df_sorted[1,]
  return(paste(bestModel$model, bestModel$G))
}


#Finds index of desired model through modelName in
model$BIC, with model given also
#modelName: Name of model for which
index will be found (format: "EVE␣2")
#model: Model object from using Mclust
#Returns: Index of modelName in model$BIC,
such that model$BIC[index]
#returns BIC value for specified modelName
findModelIndexBIC = function(modelName, model) {
  modelNames2D = c("EII","VII","EEI","VEI","EVI","VVI","EEE","VEE",
  "EVE","VVE","EEV","VEV","EVV","VVV")
  data = model$data
  modelSplit = strsplit(modelName, "␣")
  modelType = modelSplit[[1]][1]
  clusters = as.numeric(modelSplit[[1]][2])
  clustersConsidered = attr(model$BIC, "G")
  #Determine row where desired BIC exists in model$BIC
  row = match(clusters, clustersConsidered)
  if (is.na(row)) {
    print("Error:␣Amount␣of␣clusters␣of␣to␣be␣found␣model␣given␣not␣considered␣by␣model")
    return()
  }
  dataDim = ncol(data)
  #Determine column where desired BIC exists in model$BIC, which depends on dimension data
  if (dataDim == 1) {
    col = match(modelType, modelNames1D)
  }
  else if (dataDim == 2) {
    col = match(modelType, modelNames2D)
  }
  else {
    print("Error:␣Data␣dimension␣out␣of␣implemented␣options")
    return()
  }
  #When accessing indices in model$BIC,
  return((col - 1) * length(clustersConsidered) + row)
}

#Creates data using specified 2-component normal parameters,
and returns combined data
#xmu1 = x-mean component 1
#ymu1 = y-mean component 1
#xmu2 = x-mean component 2
#ymu2 = y-mean component 2
#sigma1 = covariance matrix component 1 (syntax: matrix (c(a,b,c,d), nrow = 2))
#sigma2 = covariance matrix component 2 (syntax: matrix (c(a,b,c,d), nrow = 2))
#n1 = data points component 1
#n2 = data points component 2
combineData2Norm = function(xmu1, ymu1, xmu2, ymu2, sigma1, sigma2, n1, n2) {
  mu1 = c(xmu1, ymu1)
  mu2 = c(xmu2, ymu2)
```

```
  #Cluster 1
  clus1 = mvrnorm(n = n1, mu = mu1, Sigma = sigma1)
  #Cluster 2
  clus2 = mvrnorm(n = n2, mu = mu2, Sigma = sigma2)

  #Combine
  data = rbind(clus1, clus2)
  return(data)
}
```

C.2. 1D-mean shift experiment

C.2.1. Examples

```
## Creates Figure 3.1
xSeq= seq(-3,9, length.out = 200)
ySeq= seq(-2.5, 2.5, length.out = 200)
grid = expand.grid(xSeq = xSeq, ySeq = ySeq)
coords = cbind(grid$xSeq, grid$ySeq)

covMatrix = matrix(c(1, 0.5,
             0.5, 1), nrow = 2)

mean1 = c(0,0)

mean2 = c(6,0)


grid1 = grid
grid1$z = dmvnorm(coords, mean1, covMatrix)
grid2 = grid
grid2$z = dmvnorm(coords, mean2, lambda2*D%*%A%*%t(D))

ggplot() +
  geom_contour(
    data = grid1,
    aes(xSeq, ySeq, z = z),
    color = "red"
  ) +
  geom_contour(
    data = grid2,
    aes(xSeq, ySeq, z = z),
    color = "blue"
  ) +
  coord_fixed() +
  coord_cartesian(xlim = c(-3, 9), ylim = c(-2.5, 2.5)) +
  labs(
    x = "",
    y = ""
  )

## Creates Figure 3.5
data = combineData2Norm(xmu1 = 0, ymu1 = 0,
                        xmu2 = -1.7, ymu2 = 0,
                        sigma1 = matrix(c(1, 0.5,
                                          0.5, 1), nrow = 2),
```

```
                        sigma2 = matrix(c(1, 0.5,
                                          0.5, 1), nrow = 2),
                        n1 = 500, n2 = 500)
pointModel = Mclust(data, G = c(1,2,3,4,5), verbose = FALSE)
# Preffered components by Mclust is 1, so appropriate example
#Generative EEE distributions for overlay

xSeq= seq(-14,14, length.out = 200)
ySeq= seq(-14,14, length.out = 200)
grid = expand.grid(xSeq = xSeq, ySeq = ySeq)
coords = cbind(grid$xSeq, grid$ySeq)


covMatrix = matrix(c(1, 0.5,
             0.5, 1), nrow = 2)


grid1 = grid
grid1$z = dmvnorm(coords, c(0,0), covMatrix)
grid2 = grid
grid2$z = dmvnorm(coords, c(-1.7,0), covMatrix)

ggplot(data.frame(
  x = data[,1],
  y = data[,2]
), aes(x, y)) +
  geom_point(alpha = 0.4, size = 1) +
  geom_contour(
    data = grid1,
    aes(xSeq, ySeq, z = z),
    color = "red"
  ) +
  geom_contour(
    data = grid2,
    aes(xSeq, ySeq, z = z),
    color = "blue"
  ) +
  coord_fixed() +
  labs(
    x = "",
    y = ""
  ) +
  theme(
    axis.text.x = element_blank(),
    axis.text.y = element_blank()
  )
```

C.2.2. Result figures

```
## Creates Figure 3.2
#Desired model: "EEE 2"
(ellipsoidal, equal volume, shape, and orientation)

#ppn -> points per number; (for example amount of points for interval [0,1))
ppn = 10
xLow = -6
```

```
xHigh = 6
points = seq(xLow,xHigh,by = 1/ppn)
numPoints = length(points)

simulationsPerPoint = 10
ranksBIC = vector("list", length(points))

target = "EEE␣2"

run1DShift = function(i) {
  point = points[i]
  res = numeric(simulationsPerPoint)
  for (sim in 1:simulationsPerPoint) {
    data = combineData2Norm(xmu1 = 0, ymu1 = 0,
                            xmu2 = point, ymu2 = 0,
                            sigma1 = matrix(c(1, 0.5,
                                              0.5, 1), nrow = 2),
                            sigma2 = matrix(c(1, 0.5,
                                              0.5, 1), nrow = 2),
                            n1 = 500, n2 = 500)
    #Note: For computational ease, only 1,2,3,4 and 5 component models are considered
    pointBICs = mclustBIC(data, G = c(1,2,3,4,5), verbose = FALSE)
    rank = modelRankingBIC(target, pointBICs)
    res[sim] = rank
  }
  return(res)
}

set.seed(12345)
plan(multisession, workers = 4)
ranks = future_lapply(1:numPoints,
                      run1DShift,
                      future.seed = TRUE)

######

# Plot of empirical mean + SD bands
pointMeans = sapply(ranks, mean)
pointSds = sapply(ranks, sd)

plotDataFrame = data.frame(x = points,
                           pointMean = pointMeans,
                           pointSd = pointSds)

ggplot(plotDataFrame, aes(x = x, y = pointMean)) +
  geom_ribbon(aes(ymin = pointMean - pointSd,
                  ymax = pointMean + pointSd),
              alpha = 0.2) +
  geom_line(linewidth = 1) +
  geom_point(size = 2) +
  geom_hline(yintercept = 1,
             linetype = "dashed",
             color = "red") +
  geom_vline(xintercept = 0,
             linetype = "dashed",
             color = "red") +
```

```
  labs(
    x = "Horizontal␣mean␣shift",
    y = "Rank",
    title = "1D␣shift␣empirical␣mean␣shift␣+␣SD␣band"
  ) +
  theme_minimal()

## Creates Figure 3.3
#ppn -> points per number; (for example amount of points for interval [0,1))
ppn = 10
xLow = -3
xHigh = 3
points = seq(xLow,xHigh,by = 1/ppn)
numPoints = length(points)

simulationsPerPoint = 10
ranks = vector("list", length(points))

target = "EEE␣2"

run1DShift = function(i) {
  point = points[i]
  res = character(simulationsPerPoint)
  for (sim in 1:simulationsPerPoint) {
    data = combineData2Norm(xmu1 = 0, ymu1 = 0,
                            xmu2 = point, ymu2 = 0,
                            sigma1 = matrix(c(1, 0.5,
                                              0.5, 1), nrow = 2),
                            sigma2 = matrix(c(1, 0.5,
                                              0.5, 1), nrow = 2),
                            n1 = 500, n2 = 500)
    #Note: For computational ease, only 1,2,3,4 and 5 component models are considered
    pointBICs = mclustBIC(data, G = c(1,2,3,4,5), verbose = FALSE)
    res[sim] = favouredModelBIC(pointBICs)
  }
  return(res)
}

set.seed(12345)
plan(multisession, workers = 4)
ranks = future_lapply(1:numPoints,
                      run1DShift,
                      future.seed = TRUE)

######
# Create 100%-stacked bar chart
split_ranks = lapply(ranks, function(group) {
  strsplit(group, "␣")
})

modelNames = lapply(split_ranks, function(group) {
  sapply(group, `[`, 1)
})

components = lapply(split_ranks, function(group) {
  as.numeric(sapply(group, `[`, 2))
```

```
})


rankDF <- map_dfr(seq_along(modelNames), function(i) {
  tibble(
    group = xLow + (i-1) * 1/ppn,
    value = modelNames[[i]]
  )
})

rankDFProps <- rankDF %>%
  count(group, value) %>%
  group_by(group) %>%
  mutate(prop = n / sum(n))


ggplot(rankDFProps, aes(x = factor(group), y = prop, fill = value)) +
  geom_col() +
  ylab("Proportion") +
  xlab("Horizontal mean shift") +
  scale_fill_viridis_d() +
  scale_y_continuous(labels = scales::percent) +
  scale_x_discrete(
    breaks = as.character(seq(-3,3, by = 0.5))
  )

## Creates Figure 3.4
#ppn -> points per number; (for example amount of points for interval [0,1))
ppn = 10
xLow = -3
xHigh = 3
points = seq(xLow,xHigh,by = 1/ppn)
numPoints = length(points)

simulationsPerPoint = 10
ranks = vector("list", length(points))

target = "EEE 2"

run1DShift = function(i) {
  point = points[i]
  res = character(simulationsPerPoint)
  for (sim in 1:simulationsPerPoint) {
    data = combineData2Norm(xmu1 = 0, ymu1 = 0,
                            xmu2 = point, ymu2 = 0,
                            sigma1 = matrix(c(1, 0.5,
                                              0.5, 1), nrow = 2),
                            sigma2 = matrix(c(1, 0.5,
                                              0.5, 1), nrow = 2),
                            n1 = 500, n2 = 500)
    #Note: For computational ease, only 1,2,3,4 and 5 component models are considered
    pointBICs = mclustBIC(data, G = c(1,2,3,4,5), verbose = FALSE)
    res[sim] = favouredModelBIC(pointBICs)
  }
  return(res)
}
```

```
set.seed(12345)
plan(multisession, workers = 4)
ranks = future_lapply(1:numPoints,
                      run1DShift,
                      future.seed = TRUE)

######
# Create 100%-stacked bar chart
split_ranks = lapply(ranks, function(group) {
  strsplit(group, "␣")
})

modelNames = lapply(split_ranks, function(group) {
  sapply(group, '[', 1)
})

components = lapply(split_ranks, function(group) {
  as.numeric(sapply(group, '[', 2))
})

rankDF <- map_dfr(seq_along(components), function(i) {
  tibble(
    group = xLow + (i-1) * 1/ppn,
    value = components[[i]]
  )
})

rankDFProps <- rankDF %>%
  count(group, value) %>%
  group_by(group) %>%
  mutate(prop = n / sum(n)) %>%
  ungroup() %>%
  mutate(value = factor(value))

ggplot(rankDFProps, aes(x = factor(group), y = prop, fill = value)) +
  geom_col() +
  ylab("Proportion") +
  xlab("Horizontal␣mean␣shift") +
  scale_fill_viridis_d() +
  scale_y_continuous(labels = scales::percent) +
  scale_x_discrete(
    breaks = as.character(seq(-3,3, by = 0.5))
  )
```

C.3. Orientation shift experiment

C.3.1. Examples

```
## Creates Figure 3.6
xSeq= seq(-15, 15, length.out = 200)
ySeq= seq(-10, 10, length.out = 200)
grid = expand.grid(xSeq = xSeq, ySeq = ySeq)
coords = cbind(grid$xSeq, grid$ySeq)

mean = c(0,0)
```

```
lambda = 4

A = matrix(c(10,0,
             0,0.1), nrow = 2)

D2 = matrix(c(cos(pi/4), -sin(pi/4),
              sin(pi/4), cos(pi/4)), nrow = 2)

grid1 = grid
grid1$z = dmvnorm(coords, mean, lambda*A)
grid2 = grid
grid2$z = dmvnorm(coords, mean, lambda2*D2%*%A%*%t(D2))

ggplot() +
  geom_contour(
    data = grid1,
    aes(xSeq, ySeq, z = z),
    color = "red"
  ) +
  geom_contour(
    data = grid2,
    aes(xSeq, ySeq, z = z),
    color = "blue"
  ) +
  coord_fixed() +
  coord_cartesian(xlim = c(-15, 15), ylim = c(-10, 10)) +
  labs(
    x = "",
    y = ""
  )

## Creates Figure 3.9
A = matrix(c(10, 0,
             0, 0.1), nrow = 2)
lambda = 4
BaseCovMatrix = A*lambda
angle = pi *0.03

data = combineData2Norm(xmu1 = 0, ymu1 = 0,
                        xmu2 = 0, ymu2 = 0,
                        sigma1 = BaseCovMatrix,
                        sigma2 = lambda * matrix(c(cos(angle), -sin(angle),
                                                   sin(angle), cos(angle)), nrow = 2) %*% A %*%
                          matrix(c(cos(angle), sin(angle),
                                   -sin(angle), cos(angle)), nrow = 2),
                        n1 = 500, n2 = 500)
#Note: For computational ease, only 1,2,3,4 and 5 component models are considered
pointModel = Mclust(data, G = c(1,2,3,4,5), verbose = FALSE)
# Preffered component by Mclust is 1, so appropriate example
#Generative EEV distributions for overlay

xSeq= seq(-14,14, length.out = 200)
ySeq= seq(-14,14, length.out = 200)
grid = expand.grid(xSeq = xSeq, ySeq = ySeq)
coords = cbind(grid$xSeq, grid$ySeq)
```

```
grid1 = grid
grid2 = grid
grid1$z = dmvnorm(coords, c(0,0), BaseCovMatrix)
grid2$z = dmvnorm(coords, c(0,0), lambda * matrix(c(cos(angle), -sin(angle),
                                                    sin(angle), cos(angle)), nrow = 2) %*% A %*%
                  matrix(c(cos(angle), sin(angle),
                           -sin(angle), cos(angle)), nrow = 2))

ggplot(data.frame(
  x = data[,1],
  y = data[,2]
), aes(x, y)) +
  geom_point(alpha = 0.4, size = 1) +
  geom_contour(
    data = grid1,
    aes(xSeq, ySeq, z = z),
    color = "red"
  ) +
  geom_contour(
    data = grid2,
    aes(xSeq, ySeq, z = z),
    color = "blue"
  ) +
  theme(aspect.ratio = 0.8) +
  labs(
    x = "",
    y = ""
  )
```

C.3.2. Result figures

```
## Creates Figure 3.7
#ppn -> points per number; (for example amount of points for interval [0,1))
thetaLow = 0
thetaHigh = pi
numAngles = 101
angles = seq(thetaLow,thetaHigh,length.out = numAngles)

#Create matrices for assembling covariance matrices
A = matrix(c(10, 0,
             0, 0.1), nrow = 2)
lambda = 4
BaseCovMatrix = A*lambda


simulationsPerAngle = 10
ranks = vector("list", length(angles))

target = "EEV 2"

runOrientationShift = function(i) {
  angle = angles[i]
  res = numeric(simulationsPerAngle)
  for (sim in 1:simulationsPerAngle) {
```

```
    data = combineData2Norm(xmu1 = 0, ymu1 = 0,
                            xmu2 = 0, ymu2 = 0,
                            sigma1 = BaseCovMatrix,
                            sigma2 = lambda * matrix(c(cos(angle), -sin(angle),
                                                       sin(angle), cos(angle)), nrow = 2) %*% A %*%
                                             matrix(c(cos(angle), sin(angle),
                                                      -sin(angle), cos(angle)), nrow = 2),
                            n1 = 500, n2 = 500)
    #Note: For computational ease, only 1,2,3,4 and 5 component models are considered
    pointBICs = mclustBIC(data, G = c(1,2,3,4,5), verbose = FALSE)
    rank = modelRankingBIC(target, pointBICs)
    res[sim] = rank
  }
  return(res)
}

set.seed(12345)
plan(multisession, workers = 4)
ranks = future_lapply(1:numAngles,
                      runOrientationShift,
                      future.seed = TRUE)

######

# Plot of empirical mean + SD bands
pointMeans = sapply(ranks, mean)
pointSds = sapply(ranks, sd)

plotDataFrame = data.frame(x = angles,
                           pointMean = pointMeans,
                           pointSd = pointSds)

ggplot(plotDataFrame, aes(x = x, y = pointMean)) +
  geom_ribbon(aes(ymin = pointMean - pointSd,
                  ymax = pointMean + pointSd),
              alpha = 0.2) +
  geom_line(linewidth = 1) +
  geom_point(size = 2) +
  geom_hline(yintercept = 1,
             linetype = "dashed",
             color = "red") +
  geom_vline(xintercept = 0,
             linetype = "dashed",
             color = "red") +
  geom_vline(xintercept = pi,
             linetype = "dashed",
             color = "red") +
  geom_vline(xintercept = pi/2,
             linetype = "dashed",
             color = "blue") +
  labs(
    x = "Angle␣shift",
    y = "Rank",
    title = "Angle␣shift␣empirical␣mean␣+␣SD␣band"
  ) +
  theme_minimal()
```

```
 ## Creates Figure 3.8
 #ppn -> points per number; (for example amount of points for interval [0,1))
thetaLow = 0
thetaHigh = pi
numAngles = 101
angles = seq(thetaLow,thetaHigh,length.out = numAngles)

#Create matrices for assembling covariance matrices
A = matrix(c(10, 0,
             0, 0.1), nrow = 2)
lambda = 4
BaseCovMatrix = A*lambda


simulationsPerAngle = 10
ranks = vector("list", length(angles))

target = "EEV 2"

runOrientationShift = function(i) {
  angle = angles[i]
  res = numeric(simulationsPerAngle)
  for (sim in 1:simulationsPerAngle) {
    data = combineData2Norm(xmu1 = 0, ymu1 = 0,
                            xmu2 = 0, ymu2 = 0,
                            sigma1 = BaseCovMatrix,
                            sigma2 = lambda * matrix(c(cos(angle), -sin(angle),
                                                       sin(angle), cos(angle)), nrow = 2) %*% A %*%
                              matrix(c(cos(angle), sin(angle),
                                       -sin(angle), cos(angle)), nrow = 2),
                            n1 = 500, n2 = 500)
    #Note: For computational ease, only 1,2,3,4 and 5 component models are considered
    pointBICs = mclustBIC(data, G = c(1,2,3,4,5), verbose = FALSE)
    res[sim] = favouredModelBIC(pointBICs)

  }
  return(res)
}

set.seed(12345)
plan(multisession, workers = 4)
ranks = future_lapply(1:numAngles,
                      runOrientationShift,
                      future.seed = TRUE)

######
split_ranks = lapply(ranks, function(group) {
  strsplit(group, " ")
})

modelNames = lapply(split_ranks, function(group) {
  sapply(group, `[`, 1)
})

components = lapply(split_ranks, function(group) {
  as.numeric(sapply(group, `[`, 2))
```

```
})


rankDF <- map_dfr(seq_along(modelNames), function(i) {
  tibble(
    group = thetaLow + (i-1) * pi/numAngles,
    value = modelNames[[i]]
  )
})

rankDFProps <- rankDF %>%
  count(group, value) %>%
  group_by(group) %>%
  mutate(prop = n / sum(n))


ggplot(rankDFProps, aes(x = factor(group), y = prop, fill = value)) +
  geom_col() +
  ylab("Proportion") +
  xlab("Orientation shift") +
  scale_fill_viridis_d() +
  scale_y_continuous(labels = scales::percent) +
  scale_x_discrete(breaks = c("0", "1", "2", "3"))

#Creates Figure 3.10
#ppn -> points per number; (for example amount of points for interval [0,1))
thetaLow = 0
thetaHigh = pi
numAngles = 101
angles = seq(thetaLow,thetaHigh,length.out = numAngles)

#Create matrices for assembling covariance matrices
A = matrix(c(10, 0,
             0, 0.1), nrow = 2)
lambda = 4
BaseCovMatrix = A*lambda


simulationsPerAngle = 10
ranks = vector("list", length(angles))

target = "EEV 2"

runOrientationShift = function(i) {
  angle = angles[i]
  res = numeric(simulationsPerAngle)
  for (sim in 1:simulationsPerAngle) {
    data = combineData2Norm(xmu1 = 0, ymu1 = 0,
                            xmu2 = 0, ymu2 = 0,
                            sigma1 = BaseCovMatrix,
                            sigma2 = lambda * matrix(c(cos(angle), -sin(angle),
                                                       sin(angle), cos(angle)), nrow = 2)
                                                       %*% A %*%
                              matrix(c(cos(angle), sin(angle),
                                       -sin(angle), cos(angle)), nrow = 2),
                            n1 = 500, n2 = 500)
```

```
    #Note: For computational ease, only 1,2,3,4 and 5 component models are considered
    pointBICs = mclustBIC(data, G = c(1,2,3,4,5), verbose = FALSE)
    res[sim] = favouredModelBIC(pointBICs)

  }
  return(res)
}

set.seed(12345)
plan(multisession, workers = 4)
ranks = future_lapply(1:numAngles,
                      runOrientationShift,
                      future.seed = TRUE)

######
split_ranks = lapply(ranks, function(group) {
  strsplit(group, "␣")
})

modelNames = lapply(split_ranks, function(group) {
  sapply(group, `[`, 1)
})

components = lapply(split_ranks, function(group) {
  as.numeric(sapply(group, `[`, 2))
})

rankDF <- map_dfr(seq_along(components), function(i) {
  tibble(
    group = xLow + (i-1) * 1/ppn,
    value = components[[i]]
  )
})

rankDFProps <- rankDF %>%
  count(group, value) %>%
  group_by(group) %>%
  mutate(prop = n / sum(n)) %>%
  ungroup() %>%
  mutate(value = factor(value))

ggplot(rankDFProps, aes(x = factor(group), y = prop, fill = value)) +
  geom_col() +
  ylab("Proportion") +
  xlab("Horizontal␣mean␣shift") +
  scale_fill_viridis_d() +
  scale_y_continuous(labels = scales::percent) +
  scale_x_discrete(
    breaks = as.character(seq(-3,3, by = 0.5))
  )
```

C.4. Volume shift experiment

C.4.1. Examples

```
## Creates Figure 3.11
```

```
 xSeq= seq(-13, 13, length.out = 200)
ySeq= seq(-8, 8, length.out = 200)
grid = expand.grid(xSeq = xSeq, ySeq = ySeq)
coords = cbind(grid$xSeq, grid$ySeq)

mean = c(0,0)

lambda1 = 5
lambda2 = 0.5

A = matrix(c(10,0,
             0,0.1), nrow = 2)

D = matrix(c(cos(pi/6), -sin(pi/6),
              sin(pi/6), cos(pi/6)), nrow = 2)

grid1 = grid
grid1$z = dmvnorm(coords, mean, lambda1*D%*%A%*%t(D))
grid2 = grid
grid2$z = dmvnorm(coords, mean, lambda2*D%*%A%*%t(D))

ggplot() +
  geom_contour(
    data = grid1,
    aes(xSeq, ySeq, z = z),
    color = "red"
  ) +
  geom_contour(
    data = grid2,
    aes(xSeq, ySeq, z = z),
    color = "blue"
  ) +
  coord_fixed() +
  coord_cartesian(xlim = c(-13, 13), ylim = c(-8, 8)) +
  labs(
    x = "",
    y = ""
  )
```

C.4.2. Result figures

```
## Creates Figure 3.12
#ppn -> points per number; (for example amount of points for interval [0,1))
lambdaLow = 0.1
lambdaHigh = 22.5
interval = 0.1
points = seq(lambdaLow,lambdaHigh,by = interval)
numPoints = length(points)

A = matrix(c(10, 0,
             0, 0.1), nrow = 2)
D = matrix(c(cos(pi/6), sin(pi/6),
                    sin(pi/6), cos(pi/6)), nrow = 2)
covMatrix = D %*% A %*% t(D)
```

```
simulationsPerPoint = 10
ranks = vector("list", length(points))

target = "VEE␣2"

runVolumeShift = function(i) {
  point = points[i]
  res = numeric(simulationsPerPoint)
  for (sim in 1:simulationsPerPoint) {
    data = combineData2Norm(xmu1 = 0, ymu1 = 0,
                            xmu2 = 0, ymu2 = 0,
                            sigma1 = 5*covMatrix,
                            sigma2 = point*covMatrix,
                            n1 = 500, n2 = 500)
    #Note: For computational ease, only 1,2,3,4 and 5 component models are considered
    pointBICs = mclustBIC(data, G = c(1,2,3,4,5), verbose = FALSE)
    res[sim] = modelRankingBIC(target, pointBICs)
  }
  return(res)
}

set.seed(12345)
plan(multisession, workers = 4)
ranks = future_lapply(1:numPoints,
                      runVolumeShift,
                      future.seed = TRUE)
######

# Plot of empirical mean + SD bands
pointMeans = sapply(ranks, mean)
pointSds = sapply(ranks, sd)

plotDataFrame = data.frame(x = points,
                           pointMean = pointMeans,
                           pointSd = pointSds)

ggplot(plotDataFrame, aes(x = x, y = pointMean)) +
  geom_ribbon(aes(ymin = pointMean - pointSd,
                  ymax = pointMean + pointSd),
              alpha = 0.2) +
  geom_line(linewidth = 1) +
  geom_point(size = 2) +
  geom_hline(yintercept = 1,
             linetype = "dashed",
             color = "red") +
  geom_vline(xintercept = 5,
             linetype = "dashed",
             color = "red") +
  geom_vline(xintercept = 0,
             linetype = "dashed",
             color = "red") +
  labs(
    x = "Volume␣shift",
    y = "Rank",
    title = "Volume␣shift␣(same␣position)␣empirical␣mean␣+␣SD␣band"
  ) +
```

```
  theme_minimal()

## Creates Figure 3.13
#ppn -> points per number; (for example amount of points for interval [0,1))
lambdaLow = 0.1
lambdaHigh = 22.5
interval = 0.1
points = seq(lambdaLow,lambdaHigh,by = interval)
numPoints = length(points)

A = matrix(c(10, 0,
             0, 0.1), nrow = 2)
D = matrix(c(cos(pi/6), sin(pi/6),
             sin(pi/6), cos(pi/6)), nrow = 2)
covMatrix = D %*% A %*% t(D)


simulationsPerPoint = 10
ranks = vector("list", length(points))

target = "VEE 2"

runVolumeShift = function(i) {
  point = points[i]
  res = numeric(simulationsPerPoint)
  for (sim in 1:simulationsPerPoint) {
    data = combineData2Norm(xmu1 = 0, ymu1 = 0,
                            xmu2 = 0, ymu2 = 0,
                            sigma1 = 5*covMatrix,
                            sigma2 = point*covMatrix,
                            n1 = 500, n2 = 500)
    #Note: For computational ease, only 1,2,3,4 and 5 component models are considered
    pointBICs = mclustBIC(data, G = c(1,2,3,4,5), verbose = FALSE)
    res[sim] = favouredModelBIC(pointBICs)
  }
  return(res)
}

set.seed(12345)
plan(multisession, workers = 4)
ranks = future_lapply(1:numPoints,
                      runVolumeShift,
                      future.seed = TRUE)
######

split_ranks = lapply(ranks, function(group) {
  strsplit(group, " ")
})

modelNames = lapply(split_ranks, function(group) {
  sapply(group, `[`, 1)
})

components = lapply(split_ranks, function(group) {
  as.numeric(sapply(group, `[`, 2))
})
```

```
rankDF <- map_dfr(seq_along(modelNames), function(i) {
  tibble(
    group = lambdaLow + (i-1) * lambdaHigh/numPoints,
    value = modelNames[[i]]
  )
})

rankDFProps <- rankDF %>%
  count(group, value) %>%
  group_by(group) %>%
  mutate(prop = n / sum(n))


ggplot(rankDFProps, aes(x = factor(group), y = prop, fill = value)) +
  geom_col() +
  ylab("Proportion") +
  xlab("Volume shift") +
  scale_fill_viridis_d() +
  scale_y_continuous(labels = scales::percent) +
  scale_x_discrete(breaks = as.character(seq(0,20, by = 5)))

## Creates Figure 3.14
#ppn -> points per number; (for example amount of points for interval [0,1))
lambdaLow = 0.1
lambdaHigh = 22.5
interval = 0.1
points = seq(lambdaLow,lambdaHigh,by = interval)
numPoints = length(points)

A = matrix(c(10, 0,
             0, 0.1), nrow = 2)
D = matrix(c(cos(pi/6), sin(pi/6),
             sin(pi/6), cos(pi/6)), nrow = 2)
covMatrix = D %*% A %*% t(D)


simulationsPerPoint = 10
ranks = vector("list", length(points))

target = "VEE 2"

runVolumeShift = function(i) {
  point = points[i]
  res = numeric(simulationsPerPoint)
  for (sim in 1:simulationsPerPoint) {
    data = combineData2Norm(xmu1 = 0, ymu1 = 0,
                            xmu2 = 0, ymu2 = 0,
                            sigma1 = 5*covMatrix,
                            sigma2 = point*covMatrix,
                            n1 = 500, n2 = 500)
    #Note: For computational ease, only 1,2,3,4 and 5 component models are considered
    pointBICs = mclustBIC(data, G = c(1,2,3,4,5), verbose = FALSE)
    res[sim] = favouredModelBIC(pointBICs)
  }
```

```
  return(res)
}

set.seed(12345)
plan(multisession, workers = 4)
ranks = future_lapply(1:numPoints,
                      runVolumeShift,
                      future.seed = TRUE)
######

split_ranks = lapply(ranks, function(group) {
  strsplit(group, "␣")
})

modelNames = lapply(split_ranks, function(group) {
  sapply(group, '[', 1)
})

components = lapply(split_ranks, function(group) {
  as.numeric(sapply(group, '[', 2))
})

rankDF <- map_dfr(seq_along(components), function(i) {
  tibble(
    group = lambdaLow + (i-1) * lambdaHigh/numPoints,
    value = components[[i]]
  )
})

rankDFProps <- rankDF %>%
  count(group, value) %>%
  group_by(group) %>%
  mutate(prop = n / sum(n)) %>%
  ungroup() %>%
  mutate(value = factor(value))

ggplot(rankDFProps, aes(x = factor(group), y = prop, fill = value)) +
  geom_col() +
  ylab("Proportion") +
  xlab("Volume␣shift") +
  scale_fill_viridis_d() +
  scale_y_continuous(labels = scales::percent) +
  scale_x_discrete(
    breaks = as.character(seq(0,20, by = 5))
  )
```

D

Code used in Chapter 4

D.1. Helper functions for AIC- and copula-experiments

```
 #Determines AIC for model type given BIC, number of observations
 and number of estimated parameters
#BIC = BIC-value of model
#n = number of observations data
#k = number of estimated parameters
# NOTE: assumes BIC is in mclust convention (2 logL - k log n)
AIC = function(BIC, n, k) {
  #Determine log-likelihood from BIC, as Mclust does not store
  log-likelihood for each model considered
  logLik = (BIC + k*log(n)) / 2
  return(2*k - 2*logLik)
}

#Finds index of desired model through modelName
#Note: Currently only works for model whose data is 1- or 2-dimensional
#modelName: Name of model for which index will be found (format: "EVE 2")
#components: Column containing the numbers of components considered
#dataDim: Dimension of data used
#Returns: Index of AIC corresponding to modelName in AIC values
findModelIndexAIC = function(modelName, components, dataDim) {
  modelNames1D = c("E","V")
  modelNames2D = c("EII","VII","EEI","VEI","EVI","VVI","EEE","VEE",
  "EVE","VVE","EEV","VEV","EVV","VVV")
  modelSplit = strsplit(modelName, " ")
  modelType = modelSplit[[1]][1]
  clusters = as.numeric(modelSplit[[1]][2])
  #Determine row where desired BIC exists in model$BIC
  row = match(clusters, components)
  if (is.na(row)) {
    print("Error: Amount of clusters of to be found model given not considered by model")
    return()
  }
  #Determine column where desired BIC exists in model$BIC, which depends on dimension data
  if (dataDim == 1) {
    col = match(modelType, modelNames1D)
    if (is.na(col)) {
      print("Error: Model name given incorrect")
      return()
```

```
    }
  }
  else if (dataDim == 2) {
    col = match(modelType, modelNames2D)
    if (is.na(col)) {
      print("Error: Model name given incorrect")
      return()
    }
  }
  else {
    print("Error: Data dimension out of implemented options")
    return()
  }
  return((col - 1) * length(components) + row)
}

#Finds and returns ranking of model through given AIC values
#modelName: Name of model for which index will be found (format: "EVE 2")
#AICValues: Matrix of  values of different models
#Returns: (Absolute) ranking of model given
modelRankingAIC = function(modelName, AICVals) {
  modelSplit = strsplit(modelName, " ")
  modelType = modelSplit[[1]][1]
  clusters = as.numeric(modelSplit[[1]][2])

  df = as.data.frame(as.table(AICVals))
  colnames(df) = c("G", "model", "AIC")
  df_sorted = df[order(df$AIC), ]
  return(which(df_sorted$G == clusters & df_sorted$model == modelType))
}

#Finds and returns favoured model name given list of AIC values (format: "EVE 2")
#AICValues: Matrix of AIC values of different models
#Returns: Name of favoured model type with amount of components
favouredModelAIC = function(AICVals) {
  df = as.data.frame(as.table(AICVals))
  colnames(df) = c("G", "model", "AIC")
  df_sorted = df[order(df$AIC), ]

  bestModel = df_sorted[1,]
  return(paste(bestModel$model, bestModel$G))
}

#Creates data following copula model specified for 1D-mean shift experiment
#x: x-value of horizontal mean of one of the components (other fixed at 0)
generateCopulaData = function(x) {
  dims = 2
  obs <- c(500,500)
  RVMs <- list()
  RVMs[[1]] <- VineCopula::RVineMatrix(Matrix=matrix(c(1,2,0,2),dims,dims),
                                       family=matrix(c(0,3,0,0),dims,dims),
                                       # 3 means bivariate Clayton copula
                                       par=matrix(c(0,3,0,0),dims,dims),
                                       # parameters of the first copula
                                       par2=matrix(sample(0, dims*dims, replace=TRUE),
                                       dims,dims))
```

```
  RVMs[[2]] <- VineCopula::RVineMatrix(Matrix=matrix(c(1,2,0,2),dims,dims),
                                       family=matrix(c(0,6,0,0),dims,dims),
                                       # 6 means bivariate Joe copula
                                       par=matrix(c(0,3,0,0),dims,dims),
                                       # parameters of the second copula
                                       par2=matrix(sample(0, dims*dims, replace=TRUE),
                                       dims,dims))
  margin <- matrix(c('Normal', 'Normal', 'Normal', 'Normal'), 2, 2)
  margin_pars <- array(0, dim=c(2, 2, 2))

  #margin_pars formatting, margin_pars[i,j,k] sets the ith parameter
  of the distribution of jth marginal distribution of the kth component
  margin_pars[,1,1] <- c(0, 1)
  margin_pars[,1,2] <- c(0, 1)
  margin_pars[,2,1] <- c(x, 1)
  margin_pars[,2,2] <- c(0, 1)
  x_data = rvcmm(dims, obs, margin, margin_pars, RVMs)
  return(x_data[,1:2])
}
```

D.2. AIC 1D-mean shift experiment

```
## Creates Figure 4.1
#ppn -> points per number; (for example amount of points for interval [0,1))
  ppn = 10
  xLow = -6
  xHigh = 6
  points = seq(xLow,xHigh,by = 1/ppn)
  numPoints = length(points)

  simulationsPerPoint = 10
  ranksAIC = vector("list", length(points))

  target = "EEE 2"

  run1DShift = function(i) {
    point = points[i]
    res = numeric(simulationsPerPoint)
    for (sim in 1:simulationsPerPoint) {
      data = combineData2Norm(xmu1 = 0, ymu1 = 0,
                              xmu2 = point, ymu2 = 0,
                              sigma1 = matrix(c(1, 0.5,
                                                0.5, 1), nrow = 2),
                              sigma2 = matrix(c(1, 0.5,
                                                0.5, 1), nrow = 2),
                              n1 = 500, n2 = 500)
      #Note: For computational ease, only 1,2,3,4 and 5 component models are considered
      pointAICs = AICValues(data, c(1,2,3,4,5))
      rank = modelRankingAIC(target, pointAICs)
      res[sim] = rank
    }
    return(res)
  }

  set.seed(12345)
  plan(multisession, workers = 4)
```

```
  ranksAIC = future_lapply(1:numPoints,
                           run1DShift,
                           future.seed = TRUE)


  ###################
  pointMeans = sapply(ranks, mean)
  pointSds = sapply(ranks, sd)

  plotDataFrame = data.frame(x = points,
                             pointMean = pointMeans,
                             pointSd = pointSds)

  ggplot(plotDataFrame, aes(x = x, y = pointMean)) +
    geom_ribbon(aes(ymin = pointMean - pointSd,
                    ymax = pointMean + pointSd),
                alpha = 0.2) +
    geom_line(linewidth = 1) +
    geom_point(size = 2) +
    geom_hline(yintercept = 1,
               linetype = "dashed",
               color = "red") +
    geom_vline(xintercept = 0,
               linetype = "dashed",
               color = "red") +
    labs(
      x = "Horizontal mean shift",
      y = "Rank",
      title = "1D shift empirical mean shift + SD band"
    ) +
    theme_minimal()

## Creates Figure 4.2
#ppn -> points per number; (for example amount of points for interval [0,1))
  ppn = 10
  xLow = -6
  xHigh = 6
  points = seq(xLow,xHigh,by = 1/ppn)
  numPoints = length(points)

  simulationsPerPoint = 10
  ranks = vector("list", length(points))

  target = "EEE 2"

  run1DShiftAIC = function(i) {
    point = points[i]
    res = numeric(simulationsPerPoint)
    for (sim in 1:simulationsPerPoint) {
      data = combineData2Norm(xmu1 = 0, ymu1 = 0,
                              xmu2 = point, ymu2 = 0,
                              sigma1 = matrix(c(1, 0.5,
                                                0.5, 1), nrow = 2),
                              sigma2 = matrix(c(1, 0.5,
                                                0.5, 1), nrow = 2),
                              n1 = 500, n2 = 500)
```

```
    #Note: For computational ease, only 1,2,3,4 and 5 component models are considered
    pointAICs = AICValues(data, c(1,2,3,4,5))
    res[sim] = favouredModelAIC(pointAICs)
  }
  return(res)
}

set.seed(12345)
plan(multisession, workers = 4)
ranksAIC = future_lapply(1:numPoints,
                         run1DShiftAIC,
                         future.seed = TRUE)

#####
split_ranksAIC = lapply(ranksAIC, function(group) {
  strsplit(group, "␣")
})

modelNamesAIC = lapply(split_ranksAIC, function(group) {
  sapply(group, '[', 1)
})

componentsAIC = lapply(split_ranksAIC, function(group) {
  as.numeric(sapply(group, '[', 2))
})


rankDF <- map_dfr(seq_along(modelNames), function(i) {
  tibble(
    group = xLow + (i-1) * 1/ppn,
    value = modelNames[[i]]
  )
})

rankDFProps <- rankDF %>%
  count(group, value) %>%
  group_by(group) %>%
  mutate(prop = n / sum(n))


ggplot(rankDFProps, aes(x = factor(group), y = prop, fill = value)) +
  geom_col() +
  ylab("Proportion") +
  xlab("Horizontal␣mean␣shift") +
  scale_fill_viridis_d() +
  scale_y_continuous(labels = scales::percent) +
  scale_x_discrete(
    breaks = as.character(seq(-6,6, by = 0.5))
  )

## Creates Figure 4.3
 ppn = 10
  xLow = -6
  xHigh = 6
  points = seq(xLow,xHigh,by = 1/ppn)
  numPoints = length(points)
```

```
simulationsPerPoint = 10
ranks = vector("list", length(points))

target = "EEE␣2"

run1DShiftAIC = function(i) {
  point = points[i]
  res = numeric(simulationsPerPoint)
  for (sim in 1:simulationsPerPoint) {
    data = combineData2Norm(xmu1 = 0, ymu1 = 0,
                            xmu2 = point, ymu2 = 0,
                            sigma1 = matrix(c(1, 0.5,
                                              0.5, 1), nrow = 2),
                            sigma2 = matrix(c(1, 0.5,
                                              0.5, 1), nrow = 2),
                            n1 = 500, n2 = 500)
    #Note: For computational ease, only 1,2,3,4 and 5 component models are considered
    pointAICs = AICValues(data, c(1,2,3,4,5))
    res[sim] = favouredModelAIC(pointAICs)
  }
  return(res)
}

set.seed(12345)
plan(multisession, workers = 4)
ranksAIC = future_lapply(1:numPoints,
                         run1DShiftAIC,
                         future.seed = TRUE)

#####
split_ranksAIC = lapply(ranksAIC, function(group) {
  strsplit(group, "␣")
})

modelNamesAIC = lapply(split_ranksAIC, function(group) {
  sapply(group, '[', 1)
})

componentsAIC = lapply(split_ranksAIC, function(group) {
  as.numeric(sapply(group, '[', 2))
})

rankDF <- map_dfr(seq_along(components), function(i) {
  tibble(
    group = xLow + (i-1) * 1/ppn,
    value = components[[i]]
  )
})

rankDFProps <- rankDF %>%
  count(group, value) %>%
  group_by(group) %>%
  mutate(prop = n / sum(n)) %>%
  ungroup() %>%
  mutate(value = factor(value))
```

```
  ggplot(rankDFProps, aes(x = factor(group), y = prop, fill = value)) +
    geom_col() +
    ylab("Proportion") +
    xlab("Horizontal mean shift") +
    scale_fill_viridis_d() +
    scale_y_continuous(labels = scales::percent) +
    scale_x_discrete(
      breaks = as.character(seq(-6,6, by = 0.5))
    )

## Creates Figure 4.4
#ppn -> points per number; (for example amount of points for interval [0,1))
  ppn = 10
  xLow = -6
  xHigh = 6
  points = seq(xLow,xHigh,by = 1/ppn)
  numPoints = length(points)

  simulationsPerPoint = 10
  ranks = vector("list", length(points))

  target = "EEE 2"

  run1DShiftAIC = function(i) {
    point = points[i]
    res = numeric(simulationsPerPoint)
    for (sim in 1:simulationsPerPoint) {
      data = combineData2Norm(xmu1 = 0, ymu1 = 0,
                              xmu2 = point, ymu2 = 0,
                              sigma1 = matrix(c(1, 0.5,
                                                0.5, 1), nrow = 2),
                              sigma2 = matrix(c(1, 0.5,
                                                0.5, 1), nrow = 2),
                              n1 = 500, n2 = 500)
      #Note: For computational ease, only 1,2,3,4 and 5 component models are considered
      pointAICs = AICValues(data, c(1,2,3,4,5))
      res[sim] = favouredModelAIC(pointAICs)
    }
    return(res)
  }

  set.seed(12345)
  plan(multisession, workers = 4)
  ranksAIC = future_lapply(1:numPoints,
                           run1DShiftAIC,
                           future.seed = TRUE)

  #####
  split_ranksAIC = lapply(ranksAIC, function(group) {
    strsplit(group, " ")
  })

  modelNamesAIC = lapply(split_ranksAIC, function(group) {
    sapply(group, `[`, 1)
  })
```

```
  componentsAIC = lapply(split_ranksAIC, function(group) {
    as.numeric(sapply(group, '[', 2))
  })
#BIC-based model selection for comparison
  ppn = 10
  xLow = -6
  xHigh = 6
  points = seq(xLow,xHigh,by = 1/ppn)
  numPoints = length(points)

  simulationsPerPoint = 10
  ranksBIC = vector("list", length(points))

  target = "EEE 2"

  run1DShift = function(i) {
    point = points[i]
    res = numeric(simulationsPerPoint)
    for (sim in 1:simulationsPerPoint) {
      data = combineData2Norm(xmu1 = 0, ymu1 = 0,
                              xmu2 = point, ymu2 = 0,
                              sigma1 = matrix(c(1, 0.5,
                                                0.5, 1), nrow = 2),
                              sigma2 = matrix(c(1, 0.5,
                                                0.5, 1), nrow = 2),
                              n1 = 500, n2 = 500)
      #Note: For computational ease, only 1,2,3,4 and 5 component models are considered
      pointBICs = mclustBIC(data, G = c(1,2,3,4,5), verbose = FALSE)
      rank = favouredModelBIC(pointBICs)
      res[sim] = rank
    }
    return(res)
  }

  set.seed(12345)
  plan(multisession, workers = 4)
  ranksBIC = future_lapply(1:numPoints,
                           run1DShift,
                           future.seed = TRUE)

  split_ranksBIC = lapply(ranksBIC, function(group) {
    strsplit(group, " ")
  })
  componentsBIC = lapply(split_ranksBIC, function(group) {
    as.numeric(sapply(group, '[', 2))
  })

# Convert data for displaying MCAD
  meanComponentsAIC = lapply(componentsAIC, mean)
  meanComponentsBIC = lapply(componentsBIC, mean)

  meanAccuracyAIC = lapply(meanComponentsAIC, function(x) abs(x-2))
  meanAccuracyBIC = lapply(meanComponentsBIC, function(x) abs(x-2))

  MCAD = numeric(length(meanAccuracyAIC))
```

```
  for (i in 1:length(MCAD)) {
    MCAD[[i]] = meanAccuracyBIC[[i]] - meanAccuracyAIC[[i]]
  }

  plot(MCAD)

  # Create plot displaying MCAD
  MCADVector = unlist(MCAD)
  xAxis = seq(xLow, xHigh,length.out = length(MCADVector))
  yMax = max(abs(MCADVector))
  plot(xAxis, MCADVector,
       type = "l",
       lwd = 2,
       xlab = "Horizontal mean shift",
       ylab = "MCAD",
       main = "MCAD of 1D-mean shift",
       ylim = c(-yMax, yMax))
  abline(h = 0, col = "red", lty = 2)
```

D.3. AIC volume shift experiment

```
## Creates Figure 4.5
lambdaLow = 0.1
  lambdaHigh = 22.5
  interval = 0.1
  points = seq(lambdaLow,lambdaHigh,by = interval)
  numPoints = length(points)

  A = matrix(c(10, 0,
               0, 0.1), nrow = 2)
  D = matrix(c(cos(pi/6), sin(pi/6),
               sin(pi/6), cos(pi/6)), nrow = 2)
  covMatrix = D %*% A %*% t(D)


  simulationsPerPoint = 10
  ranksAIC = vector("list", length(points))

  target = "VEE 2"

  runVolumeShiftAIC = function(i) {
    point = points[i]
    res = numeric(simulationsPerPoint)
    for (sim in 1:simulationsPerPoint) {
      data = combineData2Norm(xmu1 = 0, ymu1 = 0,
                              xmu2 = 0, ymu2 = 0,
                              sigma1 = 5*covMatrix,
                              sigma2 = point*covMatrix,
                              n1 = 500, n2 = 500)
      #Note: For computational ease, only 1,2,3,4 and 5 component models are considered
      pointAICs = AICValues(data, c(1,2,3,4,5))
      res[sim] = modelRankingAIC(target, pointAICs)
    }
    return(res)
  }
```

```
  set.seed(12345)
  plan(multisession, workers = 4)
  ranksAIC = future_lapply(1:numPoints,
                           runVolumeShiftAIC,
                           future.seed = TRUE)
  #####
  # Plot of empirical mean + SD bands
  pointMeans = sapply(ranksAIC, mean)
  pointSds = sapply(ranksAIC, sd)

  plotDataFrame = data.frame(x = points,
                             pointMean = pointMeans,
                             pointSd = pointSds)

  ggplot(plotDataFrame, aes(x = x, y = pointMean)) +
    geom_ribbon(aes(ymin = pointMean - pointSd,
                    ymax = pointMean + pointSd),
                alpha = 0.2) +
    geom_line(linewidth = 1) +
    geom_point(size = 2) +
    geom_hline(yintercept = 1,
               linetype = "dashed",
               color = "red") +
    geom_vline(xintercept = 5,
               linetype = "dashed",
               color = "red") +
    labs(
      x = "Volume␣shift",
      y = "Rank",
      title = "Volume␣shift␣(same␣position)␣empirical␣mean␣+␣SD␣band"
    ) +
    theme_minimal()

## Creates Figure 4.6
lambdaLow = 0.1
  lambdaHigh = 22.5
  interval = 0.1
  points = seq(lambdaLow,lambdaHigh,by = interval)
  numPoints = length(points)

  A = matrix(c(10, 0,
               0, 0.1), nrow = 2)
  D = matrix(c(cos(pi/6), sin(pi/6),
               sin(pi/6), cos(pi/6)), nrow = 2)
  covMatrix = D %*% A %*% t(D)


  simulationsPerPoint = 10
  ranksAIC = vector("list", length(points))

  target = "VEE␣2"

  runVolumeShiftAIC = function(i) {
    point = points[i]
    res = numeric(simulationsPerPoint)
    for (sim in 1:simulationsPerPoint) {
```

```
    data = combineData2Norm(xmu1 = 0, ymu1 = 0,
                            xmu2 = 0, ymu2 = 0,
                            sigma1 = 5*covMatrix,
                            sigma2 = point*covMatrix,
                            n1 = 500, n2 = 500)
    #Note: For computational ease, only 1,2,3,4 and 5 component models are considered
    pointAICs = AICValues(data, c(1,2,3,4,5))
    res[sim] = favouredModelAIC(pointAICs)
  }
  return(res)
}

set.seed(12345)
plan(multisession, workers = 4)
ranksAIC = future_lapply(1:numPoints,
                         runVolumeShiftAIC,
                         future.seed = TRUE)
#####

#barcharts
split_ranksAIC = lapply(ranksAIC, function(group) {
  strsplit(group, "␣")
})

modelNamesAIC = lapply(split_ranksAIC, function(group) {
  sapply(group, `[`, 1)
})

componentsAIC = lapply(split_ranksAIC, function(group) {
  as.numeric(sapply(group, `[`, 2))
})

rankDF <- map_dfr(seq_along(modelNamesAIC), function(i) {
  tibble(
    group = lambdaLow + (i-1) * lambdaHigh/numPoints,
    value = modelNamesAIC[[i]]
  )
})

rankDFProps <- rankDF %>%
  count(group, value) %>%
  group_by(group) %>%
  mutate(prop = n / sum(n))


ggplot(rankDFProps, aes(x = factor(group), y = prop, fill = value)) +
  geom_col() +
  ylab("Proportion") +
  xlab("Volume␣shift") +
  scale_fill_viridis_d() +
  scale_y_continuous(labels = scales::percent) +
  scale_x_discrete(breaks = seq(0,22.5, by = 2.5))

## Creates Figure 4.7
lambdaLow = 0.1
  lambdaHigh = 22.5
```

```
  interval = 0.1
  points = seq(lambdaLow,lambdaHigh,by = interval)
  numPoints = length(points)

  A = matrix(c(10, 0,
               0, 0.1), nrow = 2)
  D = matrix(c(cos(pi/6), sin(pi/6),
               sin(pi/6), cos(pi/6)), nrow = 2)
  covMatrix = D %*% A %*% t(D)


  simulationsPerPoint = 10
  ranksAIC = vector("list", length(points))

  target = "VEE 2"

  runVolumeShiftAIC = function(i) {
    point = points[i]
    res = numeric(simulationsPerPoint)
    for (sim in 1:simulationsPerPoint) {
      data = combineData2Norm(xmu1 = 0, ymu1 = 0,
                              xmu2 = 0, ymu2 = 0,
                              sigma1 = 5*covMatrix,
                              sigma2 = point*covMatrix,
                              n1 = 500, n2 = 500)
      #Note: For computational ease, only 1,2,3,4 and 5 component models are considered
      pointAICs = AICValues(data, c(1,2,3,4,5))
      res[sim] = favouredModelAIC(pointAICs)
    }
    return(res)
  }

  set.seed(12345)
  plan(multisession, workers = 4)
  ranksAIC = future_lapply(1:numPoints,
                           runVolumeShiftAIC,
                           future.seed = TRUE)
  #####

  #barcharts
  split_ranksAIC = lapply(ranksAIC, function(group) {
    strsplit(group, " ")
  })

  modelNamesAIC = lapply(split_ranksAIC, function(group) {
    sapply(group, '[', 1)
  })

  componentsAIC = lapply(split_ranksAIC, function(group) {
    as.numeric(sapply(group, '[', 2))
  })
###
  rankDF <- map_dfr(seq_along(componentsAIC), function(i) {
    tibble(
      group = lambdaLow + (i-1) * lambdaHigh/numPoints,
      value = componentsAIC[[i]]
```

```
    )
  })

  rankDFProps <- rankDF %>%
    count(group, value) %>%
    group_by(group) %>%
    mutate(prop = n / sum(n)) %>%
    ungroup() %>%
    mutate(value = factor(value))

  ggplot(rankDFProps, aes(x = factor(group), y = prop, fill = value)) +
    geom_col() +
    ylab("Proportion") +
    xlab("Volume shift") +
    scale_fill_viridis_d() +
    scale_y_continuous(labels = scales::percent) +
    scale_x_discrete(
      breaks = as.character(seq(0,22.5, by = 2.5))
    )

## Creates Figure 4.8
#ppn -> points per number; (for example amount of points for interval [0,1))
  lambdaLow = 0.1
  lambdaHigh = 22.5
  interval = 0.1
  points = seq(lambdaLow,lambdaHigh,by = interval)
  numPoints = length(points)

  A = matrix(c(10, 0,
               0, 0.1), nrow = 2)
  D = matrix(c(cos(pi/6), sin(pi/6),
               sin(pi/6), cos(pi/6)), nrow = 2)
  covMatrix = D %*% A %*% t(D)


  simulationsPerPoint = 10
  ranksBIC = vector("list", length(points))

  target = "VEE 2"

  runVolumeShiftBIC = function(i) {
    point = points[i]
    res = numeric(simulationsPerPoint)
    for (sim in 1:simulationsPerPoint) {
      data = combineData2Norm(xmu1 = 0, ymu1 = 0,
                              xmu2 = 0, ymu2 = 0,
                              sigma1 = 5*covMatrix,
                              sigma2 = point*covMatrix,
                              n1 = 500, n2 = 500)
      #Note: For computational ease, only 1,2,3,4 and 5 component models are considered
      pointBICs = mclustBIC(data, G = c(1,2,3,4,5), verbose = FALSE)
      res[sim] = favouredModelBIC(pointBICs)
    }
    return(res)
  }
```

```
  set.seed(12345)
  plan(multisession, workers = 4)
  ranksBIC = future_lapply(1:numPoints,
                           runVolumeShiftBIC,
                           future.seed = TRUE)
  ######
  # AIC
  ranksAIC = vector("list", length(points))

  target = "VEE 2"

  runVolumeShiftAIC = function(i) {
    point = points[i]
    res = numeric(simulationsPerPoint)
    for (sim in 1:simulationsPerPoint) {
      data = combineData2Norm(xmu1 = 0, ymu1 = 0,
                              xmu2 = 0, ymu2 = 0,
                              sigma1 = 5*covMatrix,
                              sigma2 = point*covMatrix,
                              n1 = 500, n2 = 500)
      #Note: For computational ease, only 1,2,3,4 and 5 component models are considered
      pointAICs = AICValues(data, c(1,2,3,4,5))
      res[sim] = favouredModelAIC(pointAICs)
    }
    return(res)
  }

  set.seed(12345)
  plan(multisession, workers = 4)
  ranksAIC = future_lapply(1:numPoints,
                           runVolumeShiftAIC,
                           future.seed = TRUE)
  #####

  split_ranksAIC = lapply(ranksAIC, function(group) {
    strsplit(group, " ")
  })

  modelNamesAIC = lapply(split_ranksAIC, function(group) {
    sapply(group, '[', 1)
  })

  componentsAIC = lapply(split_ranksAIC, function(group) {
    as.numeric(sapply(group, '[', 2))
  })


  split_ranksBIC = lapply(ranksBIC, function(group) {
    strsplit(group, " ")
  })

  componentsBIC = lapply(split_ranksBIC, function(group) {
    as.numeric(sapply(group, '[', 2))
  })
# Convert data for displaying MCAD
  meanComponentsAIC = lapply(componentsAIC, mean)
```

```
  meanComponentsBIC = lapply(componentsBIC, mean)

  meanAccuracyAIC = lapply(meanComponentsAIC, function(x) abs(x-2))
  meanAccuracyBIC = lapply(meanComponentsBIC, function(x) abs(x-2))

  MCAD = numeric(length(meanAccuracyAIC))
  for (i in 1:length(MCAD)) {
    MCAD[[i]] = meanAccuracyBIC[[i]] - meanAccuracyAIC[[i]]
  }

  plot(MCAD)

  # Create plot displaying MCAD
  MCADVector = unlist(MCAD)
  xAxis = seq(xLow, xHigh,length.out = length(MCADVector))
  yMax = max(abs(MCADVector))
  plot(xAxis, MCADVector,
       type = "l",
       lwd = 2,
       xlab = "Horizontal mean shift",
       ylab = "MCAD",
       main = "MCAD of 1D-mean shift",
       ylim = c(-yMax, yMax))
  abline(h = 0, col = "red", lty = 2)
```

D.4. Bivariate copula mixture model experiment

```
## Creates Figure 4.9
tau <- 0.5  # target dependence strength (Kendall's tau)

families <- list(
  Gaussian = normalCopula(),
  t        = tCopula(df = 4, df.fixed = TRUE),
  Frank    = frankCopula(),
  MTCJ     = claytonCopula(),    # MTCJ is the same family as Clayton
  Joe      = joeCopula(),
  Gumbel   = gumbelCopula()
)

draw_panels <- function() {
  par(mfrow = c(2, 3), mar = c(4, 4, 3, 1))
  for (name in names(families)) {
    cop   <- families[[name]]
    theta <- iTau(cop, tau)        # find the parameter matching tau = 0.5
    cop   <- setTheta(cop, theta)

    mvd <- mvdc(cop, margins = c("norm", "norm"),
                paramMargins = list(list(mean = 0, sd = 1),
                                    list(mean = 0, sd = 1)))

    contour(mvd, dMvdc, xlim = c(-3, 3), ylim = c(-3, 3),
            n.grid = 200,
            main = name,
            drawlabels = FALSE)
  }
}
```

```
# 1) Show it in RStudio's Plots pane
draw_panels()

# 2) Also save a high-res copy to disk for your thesis
png("copula_contours.png", width = 1500, height = 1000, res = 180)
draw_panels()
dev.off()

## Creates Figure 4.10
ppn = 10
  xLow = -6
  xHigh = 6
  points = seq(xLow,xHigh,by = 1/ppn)
  numPoints = length(points)

  simulationsPerPoint = 10
  ranks = vector("list", length(points))

  target = "EEE␣2"

  run1DShiftBIC = function(i) {
    point = points[i]
    res = numeric(simulationsPerPoint)
    for (sim in 1:simulationsPerPoint) {
      data = generateCopulaData(point)
      #Note: For computational ease, only 1,2,3,4 and 5 component models are considered
      pointBICs = mclustBIC(data, c(1,2,3,4,5))
      res[sim] = favouredModelBIC(pointBICs)
    }
    return(res)
  }

  set.seed(12345)
  plan(multisession, workers = 4)
  ranksBIC = future_lapply(1:numPoints,
                           run1DShiftBIC,
                           future.seed = TRUE)

  #####

  split_ranksBIC = lapply(ranksBIC, function(group) {
    strsplit(group, "␣")
  })
  componentsBIC = lapply(split_ranksBIC, function(group) {
    as.numeric(sapply(group, '[', 2))
  })

  rankDF <- map_dfr(seq_along(componentsBIC), function(i) {
    tibble(
      group = xLow + (i-1) * 1/ppn,
      value = componentsBIC[[i]]
    )
  })
  rankDFProps <- rankDF %>%
    count(group, value) %>%
```

```
    group_by(group) %>%
    mutate(prop = n / sum(n)) %>%
    ungroup() %>%
    mutate(value = factor(value))

  ggplot(rankDFProps, aes(x = factor(group), y = prop, fill = value)) +
    geom_col() +
    ylab("Proportion") +
    xlab("Horizontal mean shift") +
    scale_fill_viridis_d() +
    scale_y_continuous(labels = scales::percent) +
    scale_x_discrete(
      breaks = as.character(seq(-6,6, by = 0.5)))

## Creates Figure 4.11
x = 6
data = generateCopulaData(x)
#Note: For computational ease, only 1,2,3,4 and 5 component models are considered
model = Mclust(data, c(1,2,3,4,5))
xSeq= seq(-14,14, length.out = 200)
ySeq= seq(-14,14, length.out = 200)
grid = expand.grid(xSeq = xSeq, ySeq = ySeq)
coords = cbind(grid$xSeq, grid$ySeq)


ggplot(data.frame(
  x = data[,2],
  y = data[,1]
), aes(x, y)) +
  geom_point(alpha = 0.4, size = 1)+
  theme(aspect.ratio = 0.8) +
  labs(
    x = "",
    y = ""
  )

## Creates Figure 4.12
 ppn = 10
  xLow = -6
  xHigh = 6
  points = seq(xLow,xHigh,by = 1/ppn)
  numPoints = length(points)

  simulationsPerPoint = 10
  ranks = vector("list", length(points))

  target = "EEE 2"

  run1DShiftAIC = function(i) {
    point = points[i]
    res = numeric(simulationsPerPoint)
    for (sim in 1:simulationsPerPoint) {
      data = generateCopulaData(point)
      #Note: For computational ease, only 1,2,3,4 and 5 component models are considered
      pointAICs = AICValues(data, c(1,2,3,4,5))
      res[sim] = favouredModelAIC(pointAICs)
    }
```

```
  return(res)
}

set.seed(12345)
plan(multisession, workers = 4)
ranksAIC = future_lapply(1:numPoints,
                         run1DShiftAIC,
                         future.seed = TRUE)

#####

split_ranksAIC = lapply(ranksAIC, function(group) {
  strsplit(group, "␣")
})
componentsAIC = lapply(split_ranksAIC, function(group) {
  as.numeric(sapply(group, '[', 2))
})

rankDF <- map_dfr(seq_along(componentsAIC), function(i) {
  tibble(
    group = xLow + (i-1) * 1/ppn,
    value = componentsAIC[[i]]
  )
})

rankDFProps <- rankDF %>%
  count(group, value) %>%
  group_by(group) %>%
  mutate(prop = n / sum(n)) %>%
  ungroup() %>%
  mutate(value = factor(value))

ggplot(rankDFProps, aes(x = factor(group), y = prop, fill = value)) +
  geom_col() +
  ylab("Proportion") +
  xlab("Horizontal␣mean␣shift") +
  scale_fill_viridis_d() +
  scale_y_continuous(labels = scales::percent) +
  scale_x_discrete(
    breaks = as.character(seq(-6,6, by = 0.5)))
```

Bibliography

Akaike, Hirotugu (1974). "A New Look at the Statistical Model Identification". In: *IEEE Transactions on Automatic Control* 19.6, pp. 716–723. DOI: 10.1109/TAC.1974.1100705.

Banfield, Jeffrey D. and Adrian E. Raftery (1993). "Model-Based Gaussian and Non-Gaussian Clustering". In: *Biometrics* 49.3, pp. 803–821. DOI: 10.2307/2532201.

Celeux, Gilles and Gérard Govaert (1995). "Gaussian parsimonious clustering models". In: *Pattern Recognition* 28.5, pp. 781–793. ISSN: 0031-3203. DOI: https://doi.org/10.1016/0031-3203(94)00125-6. URL: https://www.sciencedirect.com/science/article/pii/0031320394001256.

Charig, C. R. et al. (1986). "Comparison of Treatment of Renal Calculi by Open Surgery, Percutaneous Nephrolithotomy, and Extracorporeal Shock Wave Lithotripsy". In: *British Medical Journal* 292, pp. 879–882.

Kass, Robert E. and Adrian E. Raftery (1995). "Bayes Factors". In: *Journal of the American Statistical Association* 90.430, pp. 773–795. DOI: 10.1080/01621459.1995.10476572. URL: https://doi.org/10.1080/01621459.1995.10476572.

Nelsen, Roger B. (2006). *An Introduction to Copulas*. 2nd. Springer Series in Statistics. New York: Springer.

Schwarz, Gideon (1978). "Estimating the Dimension of a Model". In: *The Annals of Statistics* 6.2, pp. 461–464. DOI: 10.1214/aos/1176344136.

Scrucca, Luca, Michael Fop, et al. (2016). "mclust 5: Clustering, classification and density estimation using Gaussian finite mixture models". In: *The R Journal* 8.1, pp. 289–317. DOI: 10.32614/RJ-2016-021. URL: https://doi.org/10.32614/RJ-2016-021.

Scrucca, Luca, Chris Fraley, et al. (2023). *Model-Based Clustering, Classification, and Density Estimation Using mclust in R*. Chapman and Hall/CRC. ISBN: 978-1032234953. DOI: 10.1201/9781003277965. URL: https://mclust-org.github.io/book/.

Simpson, Edward H. (1951). "The Interpretation of Interaction in Contingency Tables". In: *Journal of the Royal Statistical Society: Series B* 13.2, pp. 238–241.