Mathematical Instute

Master Thesis

Statistical Science for the Life and Behavioural Sciences

SNP Genotype Calling of Tetraploid Species Using Bivariate Modelling Approaches

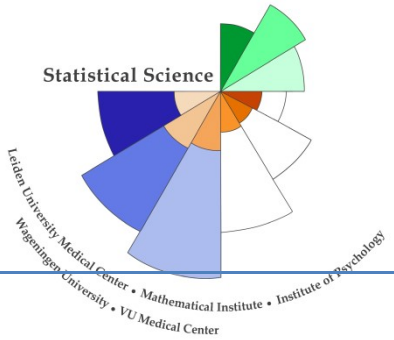
---

Author: Lennard Eijsackers

Supervisor: Gerrit Gort, Wageningen University

July 2016

Universiteit Leiden



WAGENINGEN UR
For quality of life

Abstract

In this thesis we show two methods for genotyping tetraploid species using bivariate modelling approaches. Using SNP markers we obtain a data set of fluorescent intensity signals, which are modeled to obtain an estimate for the genotypes. Previously, fitTetra was the R package of choice for analyzing this type of data and we aim to create new methods to improve the genotyping assignments of this package. The first method analyses genotype assay data by modelling the signal intensities directly, utilizing a linear regression approach to determine assignments. The second method is similar to fitTetra, modelling the ratio of signal intensities and the summed signal intensities. A regression approach is used to obtain a set of conditional means which are used to determine genotype assignments.

Contents

Introduction

In order to enhance plant breeding efficiency, genetic analysis is of great importance. One of the first and essential steps of this genetic analysis is the ability to get reliable estimates of allele dosages on a set of SNP markers, termed genotype calling. The genotyped markers are used to construct genetic maps and to perform QTL mapping, which correlate the phenotype of a plant with a genotype.

Modern genotyping assays[1] are often based on assessing two signals, one for each allele of a bi-allelic SNP marker. A specific primer binds to only one allele of the SNP, emitting a fluorescent signal of a color specific to that primer when it binds. We can use this fluorescent signal to determine the genotype at that specific marker. If the intensity of one color is higher in one SNP compared to another SNP, it may mean that in the first SNP the allele corresponding to the color is more present than in the second SNP.

While a lot of programs are available for analyzing data of diploid species, not a lot of software is developed for the analysis of tetraploid species (such as Tetrasat[2] and GenomeStudio®[3]). This is because segregation patterns are more complex for tetraploid species, as instead of 3 different possible genotypes (aa, aA and AA). The allele dosage, which is used to indicate the genotype, is defined as the number of alleles A in a gene, which in the case of diploid species ranges from nulliplex ($aa = 0$), simplex ($aA = 1$) and duplex ($AA = 2$). Tetraploid species have 5 allele dosages ranging from nulliplex ($aaaa = 0$), simplex ($aaaA = 1$), duplex ($aaAA = 2$) , triplex ($aAAA = 3$) to quadruplex ($AAAA = 4$)

In a paper by Voorrips et al (2011)[4], a method for assigning SNP allele dosages for tetraploid species, implemented in the R-package 'fitTetra', is described. The algorithm is based on fitting univariate normal mixture models with 5 components on the arcsine-square root transformed signal ratios (i.e. signal of the *a* allele compared to the total signal), one component for each of the 5 genotypes. That is, fitTetra models $= arcsin\sqrt{\frac{x}{x+y}}$, where x and y are the allele signal intensities of both the a and A alleles obtained from the genotyping assay. Several models specifying the relation between component means were proposed, each with their own set of parameters and restrictions. These models are especially of use in the case when not all 5 genotypes are present (e.g. when analyzing the offspring of two homozygous parents).

However, one of the problems that arose with fitTetra in its current state is the fact that it cannot detect null alleles. Null alleles are segregating alleles that are not detected by the genotype assay. When null alleles are present, the signal intensities and signal ratios will deviate from the expected. As an example of the difficulties null alleles present, imagine the case where we measure one SNP. Instead of the signal intensity reflecting the allele dosage, one or more alleles may not bind fluoresecent primer at all, decreasing the overall signal intensities of the alleles, meaning the signal ratio's don't accurately reflect the actual allele dosage.

To be able to detect null alleles, a method needs to be developed that includes signal intensities as well as signal ratios in its analysis. A possible solution is to model the signal intensities using a bivariate normal mixture model, as opposed to the univariate mixture model currently employed in fitTetra. We hope that we can recover the loss of signal intensities, by modelling the signal intensities directly, instead of the ratio between the signal intensities.

However we found out early on that the null alleles showed patterns that were completely different than the patterns of tetraploid species, and in order to identify them we needed an algorithm that would be able to switch between two different methods, one for identifying null alleles and one for genotyping tetraploid species. Adding an extra dimension was not enough to identify null alleles. We decided we would no longer aim to identify null alleles and instead focus on improving the current method by including an extra dimension of information. Null alleles are still mentioned as they were the original aim of creating new methods.

Another benefit of including an extra dimension of information, in the form of signal intensities, is the possibility of increased genotyping accuracy. Increased accuracy would lead to improved genotype calling and ultimately to more efficient plant breeding, and is thus a desirable property to aim for

In a presentation by Hinde (2013)[5], he proposed a method for using a bivariate normal mixture model to model the fluorescent intensity signals obtained by the genotyping assay**.** This method was specifically proposed for genotyping sugar cane, which has a variable and generally unknown ploidy level. We refer to this method as the Hinde-method**.** He proposes the usage of total least squares, a linear regression method that assumes error in both the dependent and independent variables, to calculate the means and variances of the bivariate normal mixtures. The number of regression lines is an indication for ploidy level, with each line representing a genotype and the angle of the line can be used to determine allele dosage. The means and variances are calculated using orthogonal projection of observations on the regression line.

The advantage of this approach of bivariate modelling is the fact that the error of both signal intensities is accounted for. Also, the angle between the regression lines can be used to infer allele dosage. One can place restrictions on the angles between regression lines to make sure the regression lines accurately reflect the genotypes. This is especially useful in the cases where only several genotypes appear (for example, only the simplex and triplex genotypes appear in the data), as the angle of the regression lines can be used to determine the actual genotype.

Another approach to this problem would be to adapt fitTetra to use signal intensities as well as signal ratios.[6] This means transforming fitTetra from a univariate normal mixture modelling approach to a bivariate approach. The benefit of this approach is that this potentially would allow for a similar specification of the relation between component means as proposed in the fitTetra paper[4], allowing us to obtain genotyping results when less than the expected 5 genotypes are present.

This thesis aims to provide a method for analyzing tetraploid plant species, based on modern genotyping assays. We will begin with the Hinde method before the bivariate approach based on fitTetra will be showcased. Merits and problems with both approaches will be discussed as well. We will compare the three methods (i.e. fitTetra, Hinde Method and the bivariate approach) by conducting a simulation study.

2 Preliminaries

2.1 Mixture Models

2.1.1 Mixture Models

The aim of this thesis is to create a method for genotype calling of bivariate SNP data, which includes information on both the signal ratio and signal intensity of the fluorescent substance used in the genotyping assays. In order to do so, we need a model which is able to assign different observations to different genotypes. We need to calculate the probabilities p_{ij}, which is the probability that the ith observation belongs to component *j*. A natural choice is a mixture model, which allows us to model the presence of sub-populations (the plants belonging to the different genotypes) in an overall population, without requiring the data set to identify the sub-populations to which an observation belongs.

In general, a *C* component mixture model can be described by $f(y_i|\theta_1, \theta_2, .. \theta_C, \pi_1, \pi_2, \dots \pi_C) = \sum_{j=1}^{C} \pi_j f(y_i|\theta_j)$, where

- *C* is the number of components the mixture model contains. In our case the number of components corresponds to the number of genotypes, which is five in the case of tetraploid species. In the case of tetraploid species, the genotypes are as follows: nulliplex $(aaaa)$, simplex $(Aaaa)$, duplex $(AAaa)$, triplex $(AAAa)$ and quadruplex $(AAAA)$,
- y_i is the ith observation of the response variable measured on the ith plant (i=1,..,N). In the method used in fitTetra it is the $arcsin(sqrt(ratio))$, taking the ratio of signals for a SNP on individual plant *i*.
- π_j are the mixture proportions. They are also called the prior probability of component *j*, this to distinguish them from posterior probabilities p_{ij} (the probability that the ith observation belongs to component *j*) . They are non-negative quantities that sum to one, i.e. $0 < \pi_j < 1$ and $\sum_{j=1}^{C} \pi_j = 1$. In our case they can be interpreted as the fraction of plants of the total population belonging to genotype *j*.
- θ_j is the set of parameters of the distribution associated with the corresponding mixture component *j*. It is specific for the type of distribution, which is chosen beforehand. For example, if the Gaussian distribution is chosen, the parameter vector θ_j comprises the mean and variance of mixture component *j*.
- $f(y_i|\theta_j)$ is the probability distribution of observation y_i, parameterized on the set of parameters θ_j. We will focus on the Gaussian distribution.

2.1.2 Univariate Normal Mixture Models

As mentioned in the previous section *2.1.1*, we will model the different components describing the genotypes using the Gaussian distribution. We will start with a description of the univariate normal mixture model. This is the model used by fitTetra to call the genotype of a tetraploid population.

In section *2.1.1*, a description was given for the general mixture model. Now we will give a description of the mixture model where each component is described by a univariate normal mixture model. i.e. $f(y_i|\theta_1, \theta_2, .. \theta_C, \pi_1, \pi_2, \dots \pi_C) = \sum_{j=1}^{C} \pi_j f(y_i|\theta_j)$ where

- $f(y_i|\theta_j)$ is a univariate normal density function
- θ_j is the set of parameters of the normal distribution associated with the corresponding mixture component *j*. θ_j contain the mean μ_j and variance σ_j^2 of this normal distribution.

The likelihood of observation i is $L_i(\theta) = \sum_{j=1}^{C} \pi_j f(y_i|\theta_j)$ with and $f(y_i|\theta_j) = \frac{1}{\sigma_j\sqrt{2\pi}} e^{-\frac{(y_i - \mu_j)^2}{2\sigma_j^2}}$.

2.1.3 Multivariate Normal Mixture Models

We will now translate the univariate normal mixture model described in section *2.1.2* to the multivariate case, specifically the bivariate case. This will allow us to use both the signal intensity and signal ratio to model the genotype of individual plant species.

Now the complete model of the bivariate mixture model will become $f(x_i, y_i|\theta_1, \theta_2, .. \theta_C, \pi_1, \pi_2, \dots \pi_C) = \sum_{j=1}^{C} \pi_j f(x_i, y_i|\theta_j)$. The main differences between the univariate and bivariate case are the following:

- y_i is the i[th] observation on of the y variable, in our case corresponding to a measurement of the signal ratio of an individual plant species *i*. Or in the case of the method proposed by John Hinde, it corresponds to the signal intensity corresponding to allele a.
- x_i is the i[th] observation on of the x variable, in our case corresponding to a measurement of the the sum of both signal intensities of an individual plant species *i*. Or in the case of the method proposed by John Hinde, it corresponds to the signal intensity corresponding to allele b.
- $f(x_i, y_i|\theta_j)$ is a bivariate normal density function.
- θ_j is the set of parameters of the distribution associated with the corresponding mixture component *j*. θ_j contain the mean $\mu_j = \begin{pmatrix} \mu_x \\ \mu_y \end{pmatrix}$ (leaving out *j* for simplicity of notation) and covariance matrix $\Sigma_j = \begin{pmatrix} \sigma_x^2 & \sigma_{xy} \\ \sigma_{xy} & \sigma_y^2 \end{pmatrix}$ of the bivariate normal distribution of mixture component *j*.

The density of observation i is now $L_i(\theta) = \sum_{j=1}^{C} \pi_j f(x_i, y_i|\theta_j)$ with $f(x_i, y_i|\theta_j) = \frac{1}{\sqrt{(2\pi)^2|\Sigma_j|}} e^{\left(-\frac{1}{2}(\vartheta - \mu_j)^t \Sigma_j^{-1} (\vartheta - \mu_j)\right)}$, where

- $\vartheta_i = \begin{pmatrix} x_i \\ y_i \end{pmatrix}$, the datavector of the observation *i*.

- $\mu_j = \begin{pmatrix} \mu_x \\ \mu_y \end{pmatrix}$
- $\Sigma_j = \begin{pmatrix} \sigma_x^2 & \sigma_{xy} \\ \sigma_{xy} & \sigma_y^2 \end{pmatrix}$

2.2 The EM-Algorithm for mixture models

2.2.1 The EM-Algorithm

To find the maximum likelihood estimates of the parameters of the mixture models described in the previous sections a simple numerical solution is not available., as we need to deal with the unobserved latent variable z_{ij} (which are the membership variables which tell to which component the i[th] observation belongs.) in the calculation of the likelihood. A typical approach to fit mixture models to data is the EM-algorithm, which is capable of dealing with unobserved latent variables. The acronym EM stands for Expectation-Maximization. The EM-algorithm is an iterative method for finding the maximum likelihood estimate, and is able to deal with unobserved latent variables.

The EM-algorithm is an iterative two-step algorithm alternating between the expectation step (see also section *2.2.3*), in which the expected value of the log-likelihood given the current parameter estimates is evaluated, and the maximization step (see also section *2.2.4*), in which a new set of parameters maximizing the expected log-likelihood of the previous step are found.

A property of the EM-algorithm is the fact that there is no guarantee that the EM-algorithm converges to the global maximum of the likelihood. In the case of multimodal likelihood planes, which may occur in mixture models, the EM-algorithm might converge to a local maximum. This is dependent on the starting values chosen for the EM-algorithm. Therefore it is important to choose good starting values for the EM-algorithm.

2.2.2 Formulation as an incomplete data problem

As mentioned in section *2.2.1*, the EM-algorithm is able to deal with the unobserved latent variable z_{ij}. A useful trick that allows us to apply the EM-algorithm as if we could observe the latent variable is to introduce the component-label vector z_i, composed of zero-one indicator variables, defining from which component the data y_i have arisen (see also chapter 1.9 and 2.8 of the book finite mixture modelling)[7]. This is mostly used to lay the theoretical framework for the EM-algorithm.

Let us first define the data vector y, which consists of the elements $(y_1, y_2, \ldots . y_n)$. y is assumed to contain the realized values of i independent and identically distributed random vectors $(Y_1, Y_2, \ldots . Y_i)$. The vectors Y are distributed according to the assumed distribution of the mixture model, in our case the Gaussian distribution (described in section 2.1.3)

We now define the component-label vectors $z_i = (z_{i1}, z_{i2}, \ldots\ z_{iC})$ (i=1,..,n) as the realized values of i independent and identically distributed vectors $Z_1, Z_2, \ldots\ Z_n$ distributed

as a Multinomial distribution with C categories, C being the number of genotypes, i.e. $Z_i \sim Mult(C, 1, \pi)$ $(i = 1, .. n)$. Here π is the vector of mixture proportions π_j.

We can now describe the data as an incomplete data problem, where z_{ij} is the missing part of the data and y_i is the observed part of the data. We can then describe the complete data vector as $y_c = (y, z)$ with y the observed part of the data vector and z the unobserved latent part of the data vector.

π_j is the prior probability that an observation belongs to the component j. The posterior probability, defined as the conditional probability of individual membership of class j, given response y_i, can be derived by applying Bayes rule, which gives us $P(Z_{ij} = 1|y_i) = \frac{\pi_j f(y_i|\theta_j)}{\sum_{h=1}^{C} \pi_h f(y_i|\theta_h)} = \tau_j(y_i, \varphi)$. φ is defined here as $= (\pi_1, .. \pi_j, \theta_j)$, and θ_j is the set of parameters of the distribution associated with the corresponding mixture component j. $\tau_j(y_i, \varphi)$ is the posterior probability that y_i belongs to the j^{th} component of the mixture. We can now write down the log likelihood in terms of the complete data vector y_c, $\log L_c(\varphi) = \sum_{i=1}^{n} \sum_{j=1}^{C} z_{ij}(\log(\pi_j) + \log(f(y_i|\theta_j))$.

If z_{ij} would be observable, the estimation of π_j, and other parameters would be a rather straightforward maximum likelihood problem. In effect, the m.l.e. for π_j would reduce to $\hat{\pi}_j = \sum_{j=1}^{C} \frac{z_{ij}}{n}$.

2.2.3 The E-Step

In the E-step the expected value of the complete-data log likelihood is calculated given the data y_i and the current values of the parameter φ^t, which is defined as $\varphi^t = (\pi_1, .. \pi_j, \theta_j)^{(t)}$, where t is an indication for the current iteration. $\varphi^0$is the initial specification for φ, the starting values. We can write down the E-step as $Q(\varphi|\varphi^t) = E_{\varphi^t}[\log L(\varphi^t|y)]$.

As $\log L_c(\varphi)$ is linear in the unobservable data z_{ij}, the E-step requires the calculation of Z_{ij} given the data y_i, with Z_{ij} the random variable corresponding to z_{ij}. As shown before $E_{\varphi^t}(Z_{ij} = 1|y_i) = P(Z_{ij} = 1|y_i) = \frac{\pi_j f(y_i|\theta_j)}{\sum_{h=1}^{C} \pi_h f(y_i|\theta_h)} = \tau_j(y_i, \varphi^t)$. We can now rewrite the E-step as $Q(\varphi|\varphi^t) = \sum_{i=1}^{n} \sum_{j=1}^{C} \tau_j(y_i, \varphi^t)(\log(\pi_j) + \log(f(y_i|\theta_j)))$. Essentially the E-step consist of analyzing the log-likelihood given the current parameter estimates φ^t, weighted by the posterior probability $\tau_j(y_i, \varphi^t)$.

2.2.4 The M-Step

The M-step finds the parameters φ that maximize the quantity $Q(\varphi|\varphi^t)$ to give the updated parameters φ^{t+1}. The updated estimates π_j^{t+1} for the mixing proportions π_j are calculated separately from the updated estimates θ_j^{t+1} for the parameter vector θ_j containing the parameters for the mixture components.

As shown in section *2.2.2*, the estimation of π_j would be trivial if the z_{ij} are observable, as we could directly calculate an estimate from the data available. However, since the z_{ij} are not directly available, we replace them with the posterior probability $\tau_j\ (y_i, \varphi)$ in the E-step. Estimation of π_j then becomes $\pi_j = \sum_{j=1}^{C} \frac{\tau_j\ (y_i, \varphi^t)}{n}$. In essence, each observation is weighted with the posterior probability to belong to a certain mixture component.

The updated parameters θ_j^{t+1} are found simply by taking the derivative of the log likelihood with respect to the parameters θ_j and equating it to zero, $\sum_{i=1}^{n} \sum_{j=1}^{C} \tau_j\ (y_i, \varphi^t)(\log(\pi_j) + \partial \log\left(f\left(y_i|\theta_j\right)\right)) / \partial\theta_j = 0.$

The E-step and M-step are iterated until the EM algorithm converges. As noted before, the EM algorithm might not always convergence to a global maximum.

2.2.5 The EM-Algorithm in the case of multivariate normal mixtures

In the previous sections we described the EM-algorithm in the general case, without reference to a specific mixture density. However, we already know the distribution of the mixture components, as we chose the multivariate normal model for our mixtures. In this section we describe the EM-algorithm when we know the mixture component distributions to be multivariate normal distributions.

As described in section *2.1.3*, in the case of a multivariate normal mixture model, the likelihood of thismodel given the i-th data vector $\vartheta_i = (x_i, y_i)$ can be described as $L(\varphi) = \sum_{j=1}^{C} \pi_j f\left(x_i, y_i|\theta_j\right)$ with $f\left(x_i, y_i|\theta_j\right) = \frac{1}{\sqrt{(2\pi)^2|\Sigma_j|}} e^{\left(-\frac{1}{2}\left(\vartheta_i - \mu_j\right)^t \Sigma_j^{-1} \left(\vartheta - \mu_j\right)\right)}$. The parameters θ_j consist of the mean vector of the component μ_j and the component covariance matrix Σ_j.

In the E-step, where the unobservable z_{ij} are essentially replaced by the posterior probabilities $\tau_j\ (y_i, \varphi)$, the posterior probabilities $\tau_j\ (y_i, \varphi)$ in the case of a multivariate normal mixture now become $\tau_j(\vartheta_i, \varphi) = \frac{\pi_j N(\vartheta_i|\theta_j)}{\sum_{h=1}^{C} \pi_h N(\vartheta_i|\theta_h)}$.

- The M-step, where we find the updated parameter estimates θ_j^{t+1}, is in the case of multivariate normal mixtures available in closed form. θ_j consists only of two parts, the mean of the component μ_j and the component covariance matrix Σ_j. The updated estimates for μ_j and Σ_j respectively become $\mu_j^{t+1} = \begin{pmatrix} \mu_x \\ \mu_y \end{pmatrix}$, where $\mu_y = \frac{\sum_{i=1}^{n} \tau_j\ (\vartheta_i, \varphi^t) y_i}{\sum_{i=1}^{n} \tau_j\ (\vartheta_i, \varphi^t)}$ and $\mu_x = \frac{\sum_{i=1}^{n} \tau_j\ (\vartheta_i, \varphi^t) x_i}{\sum_{i=1}^{n} \tau_j\ (\vartheta_i, \varphi^t)}$ and $\Sigma_j^{t+1} = \begin{pmatrix} \sigma_x^2 & \sigma_{xy} \\ \sigma_{yx} & \sigma_y^2 \end{pmatrix}$, where $\sigma_y^2 = \frac{\sum_{i=1}^{n} \tau_j\ (\vartheta_i, \varphi^t)(y_i - \mu_j)(y_i - \mu_j)^T}{\sum_{i=1}^{n} \tau_j\ (\vartheta_i, \varphi^t)}$, $\sigma_x^2 = \frac{\sum_{i=1}^{n} \tau_j\ (\vartheta_i, \varphi^t)(x_i - \mu_j)(x_i - \mu_j)^T}{\sum_{i=1}^{n} \tau_j\ (\vartheta_i, \varphi^t)}$ and $\sigma_{xy} = \frac{\sum_{i=1}^{n} \tau_j\ (\vartheta_i, \varphi^t)(x_i - \mu_j)(y_i - \mu_j)^T}{\sum_{i=1}^{n} \tau_j\ (\vartheta_i, \varphi^t)}$. Essentially both the mean and covariance matrix are estimated by the weighted mean and weighted covariance matrix, using the posterior probabilities $\tau_j(\vartheta, \varphi)$ as weights.

A restriction that is often applied is to take a single covariance matrix Σ for all components instead of multiple covariance matrices Σ_j. In this case, the updated estimate Σ^{t+1} is given by $\Sigma^{t+1} = \frac{\sum_{j=1}^{C}\sum_{i=1}^{n} \tau_j(\vartheta_i,\varphi^t)\Sigma_j^{t+1}}{n}$, where Σ_j^{t+1} is given by $\Sigma_j^{t+1} = \begin{pmatrix} \sigma_x^2 & \sigma_{xy} \\ \sigma_{yx} & \sigma_y^2 \end{pmatrix}$.

2.3 Total Least Squares

Since we are dealing with data on genotyping assays which contain two fluorescence signals for two alleles, we have two variables both of which are measured with error. As mentioned in the introduction, J. Hinde proposed a method of genotyping polyploid species by fitting a linear regression line through each component to obtain the parameters for that component. However, both light signals are measured with error and one of the assumptions of simple linear regression, i.e. the independent variable is measured without error, is violated. Therefore we need a method of regression that assumes error in both variables.

The regular least squares estimator $\hat{\beta}$ for the linear model $y = X\beta + \varepsilon$ minimize the sum of squared residuals $S = r^T W r$, with $r = y - X\hat{\beta}$ the residuals and W a weighting matrix. Without weighting matrix this reduces to $S = \sum_{i=1}^{n} r^2$. $X\hat{\beta}$ is the vector of fitted values of y, i.e. $\hat{y} = X\hat{\beta}$. The variance-covariance matrix of the error vector ε is Σ_y, i.e. $var(\varepsilon) = \Sigma_y$. y contains n observations and β contains m parameters. X is a matrix with $n \times m$ elements, containing functions of the independent variable x. The parameters β are estimated by setting the gradient equations equal to zero, i.e. $\frac{\delta S}{\delta \beta_j} = 0$. There are m gradient equations, called the normal equations. Solving these yields the least-squares estimator $\hat{\beta} = (X^T X)^{-1} X^T y$ or $\hat{\beta} = (X^T W X)^{-1} X^T W y$ in the case of a weighting matrix (weighted least-squares).

Total Least Squares[8] is a regression method that assumes error in both the x and y variable. Since both x and y are observed and therefore measured with error, we compute fitted values for both x and y. Therefore, we obtain residuals for both variables. Minimizing the sum of squared residuals now takes residuals of both variables into account, i.e. $S = r_x^T \Sigma_x^{-1} r_x + r_y^T \Sigma_y^{-1} r_y$. Here Σ_x^{-1} and Σ_y^{-1} are the inverse of the covariance matrices for x and y respectively and r_x and r_y are the residuals for x and y respectively. These residuals are defined as $r_x = x - \hat{x}$ and $r_y = y - \hat{y}$, i.e. the observed value minus the fitted value. These residuals can clearly not be independent, as the fitted values of one variable depend on the other, so they must be constrained by some function. This yields the objective function to be minimized to be $F = \Delta y - \frac{\delta f}{\delta r_x} - \frac{\delta f}{\delta r_x} - X\Delta\hat{\beta} = 0$. Where f is the model function with $f = (x + r_x)\beta = y + r_y$. This means that the residuals for both the x and y variables are taken into account when calculating the regression parameter. To minimize the function F subject to the a set of constraints we need Lagrange multipliers. This yields the following solution for the parameters $\beta = (X^T \Sigma^{-1} X)^{-1} X^T \Sigma^{-1} y$, where Σ is the variance-

covariance matrix which takes both the dependent variable x and the independent variable y into account, which is given by $\Sigma = \left(-\frac{\delta f}{\delta r_x}\right)\Sigma_x\left(-\frac{\delta f}{\delta r_x}^T\right) + (-\frac{\delta f}{\delta r_y})\Sigma_y(-\frac{\delta f}{\delta r_y}^T)$.

Generally, no closed form solution is available for the Total Least Squares method, except for some special cases. Algorithms used to find a Total Least Squares solution generally involve the use of a singular value decomposition of the data matrix.

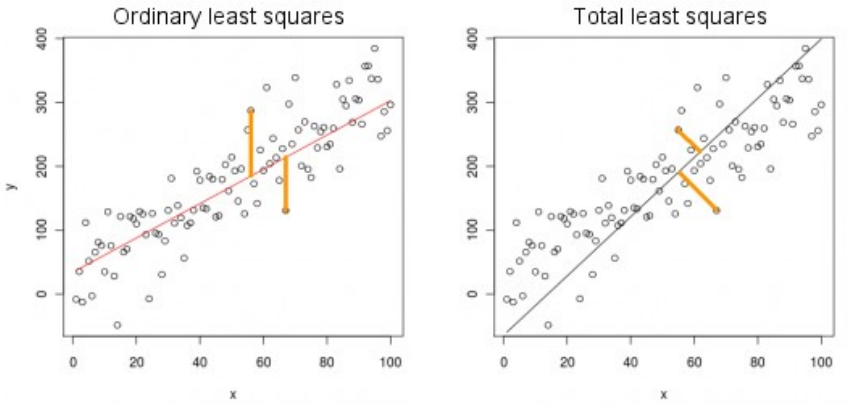


Figure 2.3.1 Visualizing the different residuals for ordinary Least Squares and Total Least Squares assuming equal variance.

2.4 Clustering

The EM-algorithm needs starting values of the parameters to start the iterations towards hopefully the maximum likelihood estimates. An obvious choice to get starting values for the parameters when dealing with mixture models is a clustering method, since we are essentially dividing the data into five clusters, namely the five different genotypes. Several clustering methods exist, all with their own set of parameters and restrictions. We need to choose a clustering method which allows us to obtain a set of starting values of parameters. For this thesis we did not look into the best method for obtaining a set of starting values, although for our purposes two methods stood: K-means clustering, as it is partitions observations into a cluster based on a cluster center, and model based clustering, such as the previously mentioned normal mixture models. Model based clustering is an approach which is similar to the approach we are applying in this thesis, modelling the data using normal mixture models. However, we allow for different types of restrictions than the regular model based clustering algorithms, such as Mclust.[9] The regular model based clustering algorithms might still be useful for finding starting values. We will briefly mention K-means clustering. For information on normal mixture models, look back at chapter *2.1*.

2.4.1 K-means clustering

K-means clustering partitions observations into clusters based on the distance to a chosen cluster mean. Therefore it is a useful method for obtaining starting parameters for the EM-algorithm, as we obtain both a cluster center and cluster variance needed for the normal

mixture models. To determine the number of clusters *k* in the data, corresponding to the number of genotypes in the data, one can use a model selection criterion such as AIC or BIC.

The algorithm for k-means is an iterative process, similar to the EM-algorithm. It determines *k* clusters based on the distance of each observation from *k* means, and then obtains a new set of means based on the previously determined clusters, until it converges. To obtain the initial set of clusters, several methods exist. We will briefly mention two of the most common initialization methods. The Forgy method randomly assigns *k* observations as means. The Random Partitioning method assigns each observation to a cluster randomly.

In order to obtain a set of *k* clusters, the k-means algorithm tries to minimize the within-cluster sums of squares, i.e. $S = \sum_{j=1}^{k}(x - \mu_j)^2$. With μ_j is … In other words, it tries to find the observations closest to the center μ_j, and partitions those observations into a cluster based on their distance from the center.

Having obtained a set of k-clusters, the means are recalculated by simply taking the average of all observations within a cluster, i.e. $\mu_j = \frac{1}{k}\sum_{x_j \in k} x_j$. When the assignments no longer change, the k-means algorithm has converged.

2.5 Biological Background

As we are working on biological data, specifically genetic data of different plant species, we will briefly mention some important biological and genetic principles relevant to this project. We will discuss SNP's, genetics and Hardy-Weinberg Equilibrium.

2.5.1 Genetics

In this section we will briefly mention the basis of genetics and heredity. As we discuss data used in the agricultural industry, we will mostly mention it from an agricultural point of view. Genetics is the study of inheritance. It aims to combine the expression of the phenotype, the combination of an individual's traits, such as height, with the genetic makeup of that individual, its genotype. A small part of the DNA of an individual might code for its height. We call this part of the DNA a gene. This information is useful in determining which genes are beneficial to pass onto the next generation, in order to obtain more beneficial traits in the next generation of individuals.

DNA is the molecule containing the genetic information of an individual plant. DNA is made up of a chain of nucleotides, bound together by deoxyribonucleic acid. There are four nucleotides in DNA, Guanine, Cytosine, Thymine and Adenine. A small DNA sequence of three nucleotides codes for an amino-acid, and combined leads to the creation of specific proteins, which carry out specific tasks in the body and lead to the expression of specific traits. For example, a gene coding for a red color in petal leaves of a flower will lead to the creation of a protein which exhibits the red color in the petal leaves.

DNA is packaged in an organized structure inside the cell called a chromosome. Most species contain two copies of a chromosome. Those species are called diploid species, indicating that they only have two copies, but other ploïdy levels are also possible. In this specific project we will mostly deal with tetraploid species, containing four copies.

2.5.2 SNP's and Null alleles

Single nucleotide polymorphisms are a DNA sequence variation in which a single nucleotide differs between alleles. As sequencing the entire DNA strand is often time consuming and expensive, SNP's offer a useful tool in genotyping. Due to the bi-allelic nature of SNP's they are easily assayed, making them ideal for biological markers. Using SNP's as markers when correlating phenotype with genotype allows us to gain information on the position of the gene correlating to a specific trait, without having to sequence the entire DNA strand.

Several assays exist to analyze these SNP markers. In this thesis we will work with data of assays based on fluorescence signals. Specific substances are created to bind to specific SNP markers, transmitting a light signal if bound. The color of the signal differs based on the allele it's bound to, creating two signals, as SNP's are bi-allelic. The ratio of intensities of these light signals (or fraction of one intensity over the total intensity) gives an indication of allele dosage.

A possible problem that can arise with these types of data is the occurrence of null alleles, alleles not picked up by the SNP assays. This causes the ratio of the light intensities to shift, giving us misinformation on the ploïdy level, which is a serious problem in the genotyping of plant species.

Using SNP's as markers for tetraploid will give us $ng = 5$ different genotypes, namely Nulliplex, Simplex, Duplex, Triplex and Quadruplex, where Nulliplex corresponds to the genotype where the a allele is absence, namely $AAAA$, and Quadruplex corresponds to the genotype where the A allele is absence, namely $aaaa$.

2.5.3 Hardy-Weinberg Equilibrium

Hardy-Weinberg Equilibrium is a useful restriction to place on genetic models. The Hardy-Weinberg principle states that allele and genotype frequencies remain constant from generation to generation in absence of other influences. This means that for diploid species, where we have two alleles A and a with frequencies p and q, the genotype frequencies are the following, $p(AA) = p^2, p(aa) = q^2$ and $p(Aa) = 2pq$. Here $p = 1 - q$.

For tetraploid species, the genotype frequencies under Hardy-Weinberg Equilibrium become $p(AAAA) = p^4, p(AAAa) = 4p^3q, p(AAaa) = 6p^2q^2, p(Aaaa) = 4pq^3$ and $p(aaaa) = q^4$.

2.6 Data

In this thesis we used the GoldenGate dataset to be used as examples for the genotyping methods (see the Results sections, chapter 5.1.1 and chapter 5.1.2). As there is no data available to us where we are certain of the genotypes of the samples, we needed another way to compare the effectiveness of the algorithms. Therefore we did a small simulation study in order compare the algorithms. In this section, we will describe these data-sets in detail. We will also describe the simulation study.

2.6.1 Available Data-sets

The dataset we used was the Illumina GoldenGate[10] data set, containing 224 samples of potato, a tetraploid plant species. In total the data set contained 384 SNP markers. We also had access to datasets containing observations on Leek, Potato, Rose and Alstroemeria, from Infinium and Axiom arrays. These were datasets potentially containing null alleles. The main variables of interest in the dataset are the raw signal intensities of both alleles of a bi-allelic SNP marker measured by the genotyping assay, which in the dataset are indicated by X_Raw for allele a and Y_raw for allele A. Figure 5.1 showcases an example of this data, plotting the signal intensities of both alleles directly.

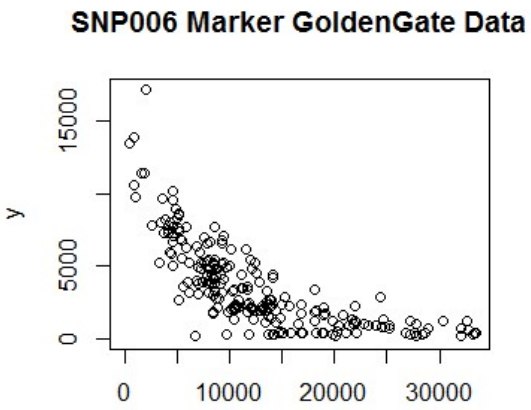


Figure 5.1. Here we show marker SNP006 of the GoldenGate Data. y represents the signal intensity of the a allele, x represents the signal intensity of the A allele.

As no method is in place yet to analyze multiple SNP's at once, which fitTetra is able to do, we handpicked a SNP marker to showcase our methods in this thesis. Most examples showcased in this thesis use this SNP as an example. The markers are blinded, so there is no indication to which marker it is. We will therefore use the name used inside of the data-set, and indicate it by SNP006

2.6.2 Simulation Study

Since the genotypes of each observation are unknown to us in the data available, we used a simulation study to compare the different methods used to genotype tetraploid species. We created a dataset based on a mixture of multivariate normals[12] using several assumptions, i.e. we sample from a distribution of the form $\sum_{j=1}^{C} \pi_j f(x_i, y_i | \theta_j)$, choosing the parameters

for the multivariate normal distribution$f\left(x_i, y_i \middle| \theta_j\right)$ and the mixture proportions π_j. The code for the simulation study is added appendix C. The actual values we used for the parameters are described in the results section in section 5.1.

We assume the genotype frequencies follow HWE, and therefore are of the form $p(AAAA) = p^4, p(AAAa) = 4p^3q, p(AAaa) = 6p^2q^2, p(Aaaa) = 4pq^3$ and $p(aaaa) = q^4$. As a value for the allele frequencies p and q we take $p = q = 0.5$. However, other values of p and q can be used. This gives us the mixture proportions π_j, $p(AAAA) = 0.0675, p(AAAa) = 0.2500, p(AAaa) = 0.3750, p(Aaaa) = 0.2500$ and $p(aaaa) = 0.0675$.

Then we sample from a Uniform (0, 1) distribution and use that to place the observation into a certain mixture component j, i.e. if a sample from the Uniform distribution would fall below 0.0675 it would be placed in the first mixture component. Once it is placed in a mixture component, we sample from the accompanying multivariate normal distribution $f\left(x_i, y_i \middle| \theta_j\right)$. The next step is to choose the parameters θ_j, containing the means μ_j and the covariance matrices Σ_j

To calculate the means $\mu_j = \binom{\mu_x}{\mu_y}$ to sample from the multivariate distribution $f\left(x_i, y_i \middle| \theta_j\right)$ we assumed a linear relationship between the allele dosage and the signal intensity, of the form $\mu_x = a_0 + a_1 x$ and $\mu_y = b_0 + b_1 y$, where a_0 and b_0 are the background signals for the fluorescence intensities of allele A and a respectively, a_1 and b_1 are the fluorescence intensities of allele A and a respectively and x and y are the allele dosages of allele A and a where $x = 4 - y$. This is similar to the constraints on μ_j fitTetra uses, adapted to the multivariate case. As with fitTetra, this model can be extended to the quadratic form, or include an additional assumption of equal background. In this case values for the fluorescence and background intensities where chosen by hand, and not based on biological data.

To calculate the covariance matrices $\Sigma_j = \begin{pmatrix} \sigma_x^2 & \sigma_{xy} \\ \sigma_{xy} & \sigma_y^2 \end{pmatrix}$, we use the general form $\Sigma_j = \begin{bmatrix} \cos\varphi_j & -\sin\varphi_j \\ \sin\varphi_j & \cos\varphi_j \end{bmatrix} \begin{bmatrix} \lambda_{1j} & 0 \\ 0 & \lambda_{2j} \end{bmatrix} \begin{bmatrix} \cos\varphi_j & \sin\varphi_j \\ -\sin\varphi_j & \cos\varphi_j \end{bmatrix}$, where λ_{1j} and λ_{2j} are the eigenvalues of covariance matrix j, and φ_j the orientation angle of covariance matrix j. The values for λ_{1j}, λ_{2j} and φ_j are chosen by hand and not based on biological data.

After a set of samples has been obtained, we check if all values are positive. This is because the genotype assays only produce positive data, and we aim to create a simulation as close to reality as possible. The negative values have been set to zero instead.

2.6.3 Transformations Applied to Data

In the case of fitTetra and the conditional method the input is the arc sine root transformed ratio of the data, i.e. $r = arcsin\sqrt{\frac{x}{x+y}}$. Here x is the signal intensity of allele A and y is the signal intensity of allele a. The second dimension used in the conditional method consists of the summed signal intensities, i.e. $d = S_a + S_b$. Figure 6.1 showcases an example of the transformed data we use for analysis.

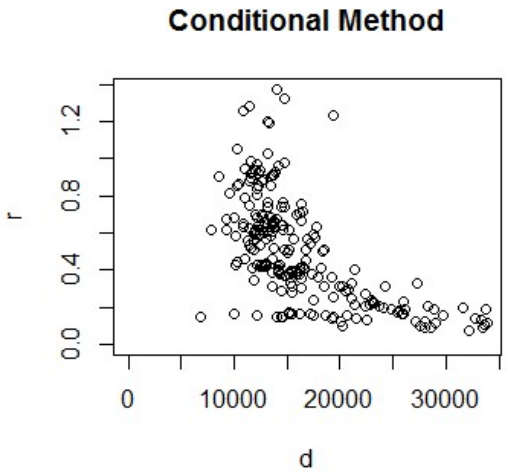


Figure 6.1. An example of the transformed data employed by the conditional method. This is marker SNP006 of the GoldenGate data.

3. Hinde Method

In this chapter we will discuss the implementation of the Hinde method, based on using both intensity signals obtained from the genotyping assay directly. We will discuss the implementation of the EM-algorithm for this specific problem step by step, starting from the initialization by the clustering algorithm. The R-code for the Hinde method can be found in appendix A. The results will be shown in chapter *5.1*. But first we will give a short description of the Hinde method itself.

The Hinde method aims to directly analyze the allele signal intensities of both the a and A alleles obtained from the genotyping assay, which are contained in the variables x and y. This means instead of only modelling the arcsine root transformed signal ratio $arc\sqrt{\frac{x}{x+y}}$ as, the method used by FitTetra, we model both x and y, giving us a two-dimensional analysis approach. The Hinde method aims to fit ng regression lines through the data and the origin, where ng is the number of genotypes. This gives us ng regression coefficients β. These regression lines can be used to determine a new set of mixture components. The angle between the regression lines gives us an indication of ploidy level, while the distance of an observation (x_i, y_i) from its orthogonal projection on regression line j gives us an indication of the likelihood of that observation to belong to mixture component j. With these likelihoods we can determine a new set of mixture component parameters, defined by the unobserved mixing proportions π_j, the within cluster variance σ_g^2 and the means μ_j.

Because both x and y are measured with error, normal regression will not suffice, as one of the least squares assumptions is violated, namely that the independent variable x is measured without error. To solve this problem, regression is done with Total Least Squares (see section 2.3).

3.1 Clustering Algorithm and Initialization

As mentioned in section 2.5.2 on the initialization of the parameters, two options were most promising in finding the starting values for the EM-algorithm, namely the K-means clustering algorithm and the R Mclust package, which uses normal mixture models. In this section we will mention the implementation of both algorithms, detailing some of the specific choices. As the Hinde method is based on regression, we will also mention the initialization we used to obtain the initial regression lines.

3.1.1 Clustering Algorithms for Initialization of the EM Algorithm

For the K-means algorithm, the R basis function kmeans was used. To determine the number of clusters of the data, which consists of the set of observations x_i and y_i, we run the kmeans function with K, the number of centers, equal to $1 \dots ng$, where ng is the number of

genotypes. The number of clusters corresponds to the number of genotypes. This means we obtain a set of ng cluster configurations, with the number of clusters in configuration g equal to g.

Now in order to determine the correct number of clusters c^* we use a more heavily penalized variant of the BIC criterion. This BIC criterion is equal to $-2 * loglikelihood + \log(n) * (7 * g)$, where n is the number of observations in the data and g is the number of clusters.[11] The extra penalization is used to avoid spurious clusters. The log-likelihood is discussed in section *3.2*.

For the initialization using normal mixture models, we use the function Mclust, available in the package mclust on CRAN. Mclust allows for the specification of different covariance structures. We use the free model, where no restrictions are placed on the covariance matrix. Mclust also uses the BIC criterion to determine the correct number of clusters. However, there is no option to implement the more heavily penalized variant we use.

Because the focus of this thesis was not on finding the ideal starting values, and because the MClust algorithm did not allow for the usage of the BIC criterion, we currently assume that the number of clusters is equal to the ploidy level we specify when initializing the EM algorithm.

3.1.2 Initialization and Starting Values

The EM-algorithm requires initial values of all parameters inside the model. This means for the Hinde method we need to obtain starting values for the regression coefficients β, the unobserved mixing proportions π_j, the within cluster variance σ_g^2 and the means μ_j. The specifics of each parameter will be discussed in their respective sections, this section deals mostly with the initialization of those parameters.

It is not possible to use weighted total least squares in order to obtain the initial regression lines, because no weights are available in the first iteration. Therefore, we used the initial clusters obtained by either the K-means or model based clustering algorithm and used those clusters as sub-datasets. These sub-datasets are then used to determine the regression coefficients, with each sub-dataset determining one regression coefficient. Essentially, we are fitting a line through the centers of these sub-datasets. This is a makeshift way to determine the initial regression lines without using weighted total least squares. For each cluster, we obtain a regression line, leading to a total of j regression lines.

After the initial regressions have been obtained, we can use those regression lines to determine the starting values for μ_j and σ_g^2 respectively. For μ_j, we first determine the orthogonal projections of each observation x_i and y_i on the j[th] regression line, namely $\widehat{x_i}$ and $\widehat{y_i}$. These orthogonal projections form $\mu_j = \begin{pmatrix} \widehat{x_i} \\ \widehat{y_i} \end{pmatrix}$. Essentially we are comparing each point to its projection on the regression line of a specific cluster.

For the within cluster variance σ_j^2, we assume that the error in both variables x and y is the same, i.e. $\sigma_x = \sigma_y$. This gives us an estimate of the cluster specific variance, which is then given by $\sigma_j^2 = \frac{\sum_{i=1}^{n}(x_i - \hat{x}_i)^2 + \sum_{i=1}^{n}(y_i - y_i)^2}{2(n-1)}$. The covariance matrix Σ_j is then given by $\Sigma_j = \begin{pmatrix} \sigma_j^2 & 0 \\ 0 & \sigma_j^2 \end{pmatrix}$. Essentially we pool the variance of both signal intensities, which is a reasonable assumption, given that they are measured in a similar way.

Since there are no posterior probabilities calculated yet, the mixture proportions are calculated as if the z_{ij} would be observable, i.e. $\pi_j = \sum_{i=1}^{n} \frac{z_{ij}}{n}$. This is estimated by simply taking the number of observations in a cluster n_c and dividing it by the total number of observations n, i.e. . $\widehat{\pi_j} = \frac{n_c}{n}$.

3.2 The E-step: Log-Likelihood and Posterior Probabilities

In this section we will discuss the evaluation of the likelihood of the Hinde method, which is the E-step of the EM-algorithm. As shown in section *2.1.3*, the likelihood of the bivariate normal mixture model is $L(\theta) = \sum_{j=1}^{C} \pi_j f(\vartheta_i|\theta_j)$, with $f(\vartheta_i|\theta_j) = \frac{1}{\sqrt{(2\pi)^2|\Sigma_j|}} e^{\left(-\frac{1}{2}(\vartheta-\mu_j)^t \Sigma_j^{-1}(\vartheta-\mu_j)\right)}$, where $\vartheta_i = \begin{pmatrix} x_i \\ y_i \end{pmatrix}$, $\mu_j = \begin{pmatrix} \mu_x \\ \mu_y \end{pmatrix}$ and $\Sigma_j = \begin{pmatrix} \sigma_x^2 & \sigma_{xy} \\ \sigma_{xy} & \sigma_y^2 \end{pmatrix}$.

With the Hinde Method, instead of using the cluster centers to evaluate the likelihood, we use the orthogonal projections of each observation on the regression lines, namely $\hat{x}_i$ and $\hat{y}_i$, with $\hat{x}_i = \frac{y_i\beta_j + x_i}{1+\beta_j^2}$ and $\hat{y}_i = \hat{x}_i\beta_j$. Using total least squares, we assume the error in both variables x and y is the same, and there is zero covariance between these variables. In reality, this is not the case. However, this assumption simplifies the mathematics needed for total least squares and in practice this tends to be a reasonable assumption. This means we can average the variance in the x and y variables giving us $\sigma_j^2 = \frac{\sum_{i=1}^{n}(x_i - \hat{x}_i)^2 + \sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{2(n-1)}$ as the within cluster variance. This gives us $L(\theta) = \sum_{j=1}^{C} \pi_j f(x_i, y_i|\theta_j)$, with $f(\vartheta_i|\theta_j) = \frac{1}{\sqrt{(2\pi)^2|\Sigma_j|}} e^{\left(-\frac{1}{2}(\vartheta-\mu_j)^t \Sigma_j^{-1}(\vartheta-\mu_j)\right)}$, where $\vartheta_i = \begin{pmatrix} x_i \\ y_i \end{pmatrix}$, $\mu_j = \begin{pmatrix} \hat{x}_i \\ \hat{y}_i \end{pmatrix}$ and $\Sigma_j = \begin{pmatrix} \sigma_j^2 & 0 \\ 0 & \sigma_j^2 \end{pmatrix}$.

We use the package mvtnorm to calculate the multivariate normal density $f(\vartheta_i|\theta_j)$. We set $\hat{x}_i$ and $\hat{y}_i$ as the mean and use the averaged covariance matrix Σ_j as variance. These densities are then multiplied with the mixture proportions π_j giving us a matrix containing the value for $\pi_j f(\vartheta_i|\theta_j)$ for each observation. Those observations are summed per row giving us the likelihood per observation $L(\theta) = \sum_{j=1}^{C} \pi_j f(\vartheta_i|\theta_j)$. To obtain the log-likelihood $\log(L(\theta))$ we need to take the sum of the log of the row sums, i.e. $\log(L(\theta)) = \sum_{i=1}^{n} \log(\sum_{j=1}^{C} \pi_j f(x_i, y_i|\theta_j))$.

To obtain the posterior probabilities of each observation to belong to cluster j we just take the matrix containing all values of $\pi_j f(\vartheta_i|\theta_j)$ for each observation and divide them by the row sums $\sum_{j=1}^{C} \pi_j f(\vartheta_i|\theta_j)$, giving us the posterior probabilities $\tau_j(\vartheta_i, \varphi) = \frac{\pi_j f(\vartheta|\theta_j)}{\sum_{h=1}^{C} \pi_h f(\vartheta|\theta_j)}$. An easy way to do this is to scale the matrix containing all values of $\pi_j f(\vartheta_i|\theta_j)$ for each observation with the row sums using the R basis function scale.

After the log-likelihood no longer changes between two iterations, i.e. $\log(L(\theta^{t+1})) = \log(L(\theta^t))$ we will assume the algorithm has converged. In R, we stop the algorithm once the change in log-likelihood between two iterations is smaller than a specific threshold.

3.3 The M-step: Updating the parameters

In this section we discuss the updating of the parameter set θ_j, containing the means μ_j, the mixture proportions π_j and the covariance matrix Σ_j. This is the M-step of the EM-algorithm, in which the parameters are updated to maximize the found log-likelihood. We start with the updating of the mixture proportions π_j.

3.3.1 Updating the mixture proportions and possible restrictions

As shown in section *2.2.4*, the mixture proportions π_j are estimated by averaging the posterior probabilities of an observation that belong to a cluster j, i.e. $\pi_j = \sum_{i=1}^{n} \frac{\tau_j(\vartheta_i, \varphi^t)}{n}$. To obtain the updated mixture proportions we take the matrix posterior probabilities containing the posterior probabilities $\tau_j(\vartheta_i, \varphi)$ and calculate the column means, yielding us estimates for π_j. This is the free estimation π_j^* of the mixture proportions π_j, as we assume no relationship.

It is possible to assume a relationship on the mixture proportions π_j. As mentioned in section *2.5.3*, when no other influences are present, we can assume genotype frequencies to stay the same across generations, resulting in the Hardy-Weinberg Equilibrium. As the mixture proportions π_j are an estimate for the genotype frequencies, we can impose the HWE equilibrium on the mixture proportions.

As mentioned in section *2.5.3*, the genotype frequencies under Hardy-Weinberg Equilibrium for tetraploid species become $p(AAAA) = p^4, p(AAAa) = 4p^3q, p(AAaa) = 6p^2q^2$, $p(Aaaa) = 4pq^3$ and $p(aaaa) = q^4$. Here p is the allele frequency of allele A and $q = 1 - p$ is the allele frequency of allele a. To calculate p, we use the free estimates π_j^* as an estimate for the genotype frequencies, and find $p = \frac{4*\pi_1^* + 3*\pi_2^* + 2*\pi_3^* + \pi_4^*}{4}$, which is an estimate for $p = \frac{4*p(AAAA) + 3*p(AAAa) + 2*p(AAaa) + p(Aaaa)}{4}$. After p is found, we can calculate the updated genotype frequencies with $\pi_1 = p^4, \pi_2 = 4p^3q, \pi_3 = 6p^2q^2, \pi_4 = 4pq^3$ and $\pi_5 = q^4$, where $q = 1 - p$.

3.3.2 Updating the means and obtaining a new set of regression lines

Instead of a centroid model, where we place an observation in a cluster based on its distance from the center of the cluster, the Hinde method calculates the distance of an observation to the regression line representing that cluster. Thus instead of having the mean be represented by $\mu_j = \begin{pmatrix} \mu_x \\ \mu_y \end{pmatrix}$, which is the case for the centroid model, the mean is the orthogonal projection of a point on the regression line representing cluster j, i.e. $\mu_j = \begin{pmatrix} \hat{x_i} \\ \hat{y_i} \end{pmatrix}$.

The first step is to find a new set of regression lines. This is achieved by performing weighted total least squares using the new updated posterior probabilities $\tau_j\ (\vartheta_i, \varphi)$. We assume the intensities x and y are measured in a similar way and therefore have similar variances .We seperatly fit the regression lines through the origin. This gives us a new set of j regression lines with regression coefficients β_j.

The next step is to find the orthogonal projections of observation (x_i, y_i) on regression line j, namely $(\hat{x_i}, \hat{y_i})$. $(\hat{x_i}, \hat{y_i})$ can be interpreted as the fitted values of an observation for total least squares. There is a set of $(\hat{x_i}, \hat{y_i})$ for each regression line j, i.e. there are 5 sets of fitted values $(\hat{x_i}, \hat{y_i})$.

The fitted values are calculated by taking the orthogonal projection of an observation (x_i, y_i) on regression line j. This gives us $\hat{x_i} = \frac{y_i\beta_j + x_i}{1 + \beta_j^2}$ and $\hat{y_i} = \hat{x_i}\beta_j$. The updated means are now found, namely $\mu_j = \begin{pmatrix} \hat{x_i} \\ \hat{y_i} \end{pmatrix}$.

For restrictions or modifications on the regression coefficients multiple multivariate total least squares is needed, in which the j regression coefficients are calculated simultaneously instead of separately. This gives different results from regular total least squares, as opposed to least squares, where multivariate multiple least squares gives the same results as least squares in most cases. We use a singular value decomposition to calculate the estimates of the regression coefficients.

Another possible restriction is to arbitrarily set the angles between the regression lines. If one would know the angles between the genotypes beforehand, this could be useful. However, this is a restriction that requires information about the data (the angles between the genotypes), which is not often available.

The third method is based on the fitTetra R-package, in which a model between allele dosage and signal intensity is used. The models for the separate intensities are $\mu_x = b_0(x) + b_1(x)a$ for the x variable and $\mu_y = b_0(y) + b_1(y)(4 - a)$ for the y variable. Here $b_0(x)$ and $b_0(y)$ are the background intensities of the variables x and y respectively, $b_1(x)$ and $b_1(y)$ are the signal intensities of the variables x and y respectively and a is the allele dosage, ranging from 0 to 4. Several other possible models are available in the fitTetra package, including a model that assumes equal background and higher order polynomials.[1]

We can estimate the parameters $b_0(x), b_1(x)$. $b_0(y)$ and $b_1(y)$ of this model and use these parameters to obtain an estimate of means $\widehat{\mu_j} = \begin{pmatrix} \mu_x \\ \mu_y \end{pmatrix}$. If we fit a regression line through the origin and $\widehat{\mu_j}$, we obtain a new β_j, through which we can calculate a new set of $\mu_j = \begin{pmatrix} \hat{x}_i \\ \hat{y}_i \end{pmatrix}$.

3.3.3 Updating the covariance matrix and restrictions on the variance

As stated before, for the covariance matrix, we assume that the variance in both x and y variables is equal, i.e. $\sigma_x = \sigma_y$. As mentioned previously, this is a reasonable assumption as both x and y represent allele signal intensities and are measured in the same way (i.e. fluorescence). In order to uphold this assumption we average the estimates of the variance of the x and y variables, obtained by calculating $\widehat{\sigma_x} = \sum_{i=1}^{n}(x_i - \hat{x}_i)^2$ and $\widehat{\sigma_y} = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2$ respectively, giving us the averaged variance $\sigma_j^2 = \frac{\sum_{i=1}^{n}(x_i - \hat{x}_i)^2 + \sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{w_i}$ where $w_i = \frac{\tau_j(\vartheta_i, \varphi)}{\sum_{i=1}^{n}(\tau_j(\vartheta_i, \varphi))}$. Notice that we use the weighted variance, i.e. each observation is weighted by the posterior probability it belongs to cluster j, obtained by the E-step. The weights are normalized. The updated covariance matrix is then given by $\Sigma_j^{t+1} = \begin{pmatrix} \sigma_j^2 & 0 \\ 0 & \sigma_j^2 \end{pmatrix}$.

A possible restriction for the variance sets the variance equal for all clusters, i.e. $\sigma_1^2 = \sigma_2^2 = \sigma_3^2 = \sigma_4^2 = \sigma_5^2 = \sigma^2$. We achieve that by calculating the average of σ_j^2, i.e. $\frac{\sum_j^C \sigma_j^2}{C}$.

4. Conditional Method

FitTetra is an R package for fitting univariate normal mixture models, modelling the arcsine root transformed ratio of signal intensities $r = arc\sqrt{\frac{x}{x+y}}$, where x and y are the allele signal intensities of both the a and A alleles, which were obtained from the genotyping assay. For the conditional bivariate method, we add an extra dimension, the summed intensities $d = x + y$. However, we still analyze it as a univariate method, but we use the extra dimension to find a conditional mean $\mu_r|\mathrm{d}$, so in a way we are using information of the second dimension. Using conditional means $\mu_r|\mathrm{d}$ we allow the means to depend on d , whereas in fitTetra the means μ_r do not have this dependence.

We start in a way, similar to the Hinde method, using a clustering method to find some initial clusters and use those clusters to find the initial regression lines. For the initial regression lines we are not able to use weighted least squares. We used the initial clusters obtained by the K-means clustering algorithm and used those clusters as sub-datasets, through which we determined the j regression coefficients.

With the regression lines we are able to determine the conditional means $\mu_r|\mathrm{d}$, and the variance coefficients $\sigma^2_{r,j}$. Together with the mixture proportions π_j we can now calculate the likelihood $L(\theta) = \sum_{j=1}^{C} \pi_j f(y_i|\theta_j)$ and use that to determine the posterior probabilities. The posterior probabilities allow us to estimate new parameters for π_j, $\mu_r|\mathrm{d}$, and $\mu_r|\mathrm{d}$. We repeat this process until the algorithm converges.

In this chapter we will discuss the implementation of the conditional method, a method similar to the one used in fitTetra utilizing an extra dimension. As in the previous chapter we will discuss the implementation of the EM-algorithm for this specific problem step by step, starting from the initialization by the clustering algorithm. The R-code for the conditional method can be found in appendix B. The results will be shown in chapter *5.2.1*. We will start with a description of the conditional method.

4.1 Clustering Algorithm and Initialization

Since we are still modeling normal mixture models, the method for clustering does not need to be changed, and we can still use either K-means or Model based clustering for the initialization of the parameters. In this section we will mention the implementation of both algorithms, detailing some of the specific choices. We will also mention the initialization of the regression line, as the conditional method uses an approach similar to the Hinde Method to obtain the conditional means mentioned in the previous section.

4.1.1 Clustering Algorithms

For the K-means algorithm, the R function kmeans was used. Instead of modelling both dimensions as was the case with the Hinde Method, we only model the first dimension,

containing the signal ratio's r, since we are not interested in the second dimension. As before we run the kmeans function with K, the number of centers, equal to$1 \dots ng$, where ng is the ploïdy level. The number of clusters is thus an indication for the number of genotypes. This means we obtain a set of ng cluster configurations, with the number of clusters in configuration g equal to g. Note that in this case, the distance from cluster center to observation is only determined by the first dimension.

The same criterion is used to determine the number of clusters, i.e. the BIC criterion that is equal to $-2 * loglikelihood + \log(n) * (7 * g)$, where n is the number of observations in the data and g is an indication for the current cluster configuration.

For the model based clustering, we use the R package Mclust. Again, we only model the first dimension. However in our testing, this approach did not give the desired results, and led to the EM-algorithm converging to local maxima. Therefore, the model based clustering approach was dropped for the conditional method.

4.1.2 Initialization and Starting Values

The EM-algorithm requires initial values of all parameters of the model. This means for the conditional method we need to obtain starting values for the regression coefficients β, the unobserved mixing proportions π_j, the variance in the y direction σ_r^2 and the conditional means $\mu_j|d$. We call these means conditional, as they contain information of the second dimension d, but the parameters do not actually include the dimension d. The specifics of each parameter will be discussed in their respective sections, this section deals mostly with the initialization of those parameters.

The initial regression lines are used to obtain the conditional means $\mu_j|d$ and the variance σ_r^2. For the conditional means $\mu_j|d$ we first determine the projection of an observation on the obtained regression lines. Since we are no longer interested in the second dimension directly, we use the fitted values $\widehat{r_i}$obtained by least squares. Similar to the Hinde Method, we use the projections as means, i.e. $\mu_j|d = \hat{r}_{i,j}$.

The variance σ_r^2 is estimated by calculating the difference between an observation r_i and its fitted value $\widehat{r_{i,J}}$, i.e. $\hat{\sigma}_r^2 = \frac{\sum_{i=1}^{n}(r_i - \widehat{r_{i,J}})^2}{n_c - 1}$. Where n_c is the number of observations in a cluster.

Since there are no posterior probabilities available in the initial phase yet, the mixture proportions are calculated as if the z_{ij} would be observable, i.e. $\pi_j = \sum_{i=1}^{n} \frac{z_{ij}}{n}$. This is estimated by simply taking the number of observations in a cluster n_c and dividing it by the total number of observations n, i.e. . $\widehat{\pi_J} = \frac{n_c}{n}$.

4.2 The E-step: Log-Likelihood and Posterior Probabilities

In this section we discuss the evaluation of the likelihood of the conditional method, the E-step of the EM-algorithm. With the conditional method, we are back to univariate modelling of the data, as we are not directly interested in the second dimension. The likelihood of the univariate normal mixture model is $L(\theta) = \sum_{j=1}^{C} \pi_j f(y_i|\theta_j)$ with $f(y_i|\theta_j) = \frac{1}{\sigma_j\sqrt{2\pi}} e^{-\frac{(y_i-\mu_j)^2}{2\sigma_j^2}}$.

In the case of the conditional method, we replace the μ_j with the fitted values $\hat{r}_{i,j}$. Here $\hat{r}_{i,j}$ are the fitted values obtained from the regression line going through each cluster. In this regard the conditional method is similar to the Hinde method, as we evaluate the likelihood of an observation to be in a cluster based on its distance from the regression line representing the cluster. This gives us a new likelihood where $f(r_i|\theta_j) = \frac{1}{\sigma_j\sqrt{2\pi}} e^{-\frac{(r_i-\hat{r}_j)^2}{2\sigma_j^2}}$.

We use the dnorm function in base R to evaluate the univariate normal density $f(r_i|\theta_j)$. We set $\hat{r}_j$ as the mean and use $\hat{\sigma}_r^2$ as variance. These densities are then multiplied with the mixture proportions π_j giving us a matrix containing the value for $\pi_j f(r_i|\theta_j)$ for each observation. Those observations are summed per row giving us the likelihood per observation $L(\theta) = \sum_{j=1}^{C} \pi_j f(r_i|\theta_j)$. To obtain the log-likelihood $\log(L(\theta))$ we need to take the sum of the log of the row sums, i.e. $\log(L(\theta)) = \sum_{i=1}^{n} \log(\sum_{j=1}^{C} \pi_j f(r_i|\theta_j))$.

The posterior probabilities are obtained in a similar way as the hinde method, using the matrix containing all values of $\pi_j f(r_i|\theta_j)$ for each observation and divide them by the row sums $\sum_{h=1}^{C} \pi_j f(y_i|\theta_j)$, giving us the posterior probabilities $\tau_j(r_i, \varphi) = \frac{\pi_j f(r_i|\theta_j)}{\sum_{h=1}^{C} \pi_h f(r_i|\theta_h)}$.

If the log-likelihood no longer changes between two iterations, i.e. $\log(L(\theta^{t+1})) = \log(L(\theta^t))$ the algorithm has converged. In R, we allow the algorithm to stop once the change in log-likelihood between two iterations is below a given small number (In our case 0.0000001).

4.3 The M-step: Updating the parameters

In this section we discuss the updating of the parameter set θ_j, containing the means μ_j, the mixture proportions π_j and the variance σ_j. This is the M-step of the EM-algorithm, in which the parameters are updated to maximize the log-likelihood. We start with the updating of the mixture proportions π_j.

4.3.1 Updating the mixture proportions and possible restrictions

To obtain the updated mixture proportions we take the matrix posterior probabilities containing the posterior probabilities $\tau_j(r_i, \varphi)$ and calculate the column means, yielding us estimates for π_j. This is the free estimation of the mixture proportions π_j, as we assume no relationship.

The relationships we can impose on the π_j are the same relationships we could impose on the Hinde method as described in section *3.3.1*, namely Hardy-Weinberg Equilibrium.

4.3.2 Updating the means

Instead of having a single value for the mean for each cluster, the conditional method aims to update the mean by calculating it conditionally on the $d = x + y$ variable, where x and y are the signal intensities of allele *a* and *A* respectively. We fit a regression line through each cluster, similar to the Hinde method, and use the fitted values obtained for each observation as the conditional mean for that observation, i.e. $\hat{\mu}_i|d = \widehat{r_{i,j}}$.

The first step is to find a new set of regression lines. The regression lines are fit by weighted least squares using the new updated posterior probabilities $\tau_j\ (r_i, \varphi)$ as weights. This gives us a new set of j regression lines with regression coefficients β_j and a set of intercepts $\mu_r|d = 0$. The set of intercepts we obtain is the mean value of the ratio when the summed intensities are 0, i.e $\mu_r|d = 0$, clearly an extrapolation.

The fitted values are obtained by projecting each observation vertically (the ratio dimension) on the regression lines, as in ordinary regression. This gives us a set of fitted values $\widehat{r_{i,j}}$, containing the fitted values for each observation projected on regression line j. This gives us the conditional means $\mu_r|\mathrm{d} = \widehat{r_i}$

As we now model the signal intensities similarly as FitTetra, we just use an extra dimension of information, we can impose the same restrictions proposed in the paper.[1] That means we impose a relationship between allele dosage and signal strength. One can model the mean signal strengths of alleles a and A as $\mu_{j,a} = a_0 + a_1 s$ and $\mu_{j,b} = b_0 + b_1 s$ respectively. Here a_0 and b_0 are the background signal intensities of alleles a and A respectively, a_1 and b_1 are the signal intensities of alleles a and A, $\mu_{j,a}$ and $\mu_{j,b}$ are the mean intensities of genotype g for a signal a and A and s is the allele dosage, ranging from nulliplex (0) to quadruplex (4). As we model the ratio's, this means we impose the restriction on the ratio's, giving us $\frac{\mu_{j,a}}{\mu_{j,a}+\mu_{j,b}} = \frac{c_1+s}{c_1+s+c_2+f(4-s)}$. Here $c_1 = \frac{a_0}{a_1}$, $c_2 = \frac{b_0}{a_1}$ and $f = \frac{b_1}{a_1}$. Using non-linear least squares we can predict a set of ratios $\frac{\mu_{j,a}}{\mu_{j,a}+\mu_{j,b}}$. These ratios are determined by the allele dosage. We then use these set of ratios as a set of points we force the j regression lines through (essentially using them as intercepts in the linear regression). This gives us the ability to predict what mixture component represents what genotype in the case when not all genotypes are present in the population, since we now have a model for determining the ploidy level.

4.3.3 Updating the variance

The variance for each cluster $\sigma^2_{r,j}$ is estimated by calculating the weighted sum of squared residuals of the regression, weighted by the posterior probabilities $\tau_j\ (\vartheta, \varphi)$, i.e. $\sigma^2_{r,j} = \frac{\sum_{i=1}^{n} \tau_j\ (r_i,\varphi)(r_i - \widehat{r_{i,j}})^2}{\sum_{i=1}^{n} \tau_j\ (r_i,\varphi)}$. To obtain the variance σ^2_r we simply take the variance of each cluster and average them, i.e.

$\sigma_r^2 = \frac{\sum_{j=1}^{C}\sum_{i=1}^{n}\tau_j\,(r_i,\varphi)(r_i-\widehat{r_{i,j}})^2}{\sum_{j=1}^{C}\sum_{i=1}^{n}\tau_j\,(r_i,\varphi)}$. Constant variance for clusters is a reasonable assumption and helps the EM-algorithm converge to the maximum instead of a local maximum, as clusters with large variance tend to be problematic.

5. Results & Discussion

In this chapter we will show results for both the Hinde method and the Conditional Method. We will showcase some data examples for both methods, showing the results obtained by both methods, starting with the Hinde Method. Then we will have a small discussion on each method after the results section. We will end with a comparison of both methods and fitTetra using a simulation study, described in chapter 2.6.

5.1 Hinde Method: Results & Discussion

In this section we will evaluate the results of the Hinde method and the restrictions described in chapter 3. Important to note is that there is no golden standard in validating the genotype assignments and we have no data available of which the genotypes are known, i.e. where another method is used to genotype the individuals. Therefore we will mostly focus on a visual analysis of the results and show the results that would have been obtained by fitTetra, mostly focusing on assignment of the observations. We also discuss the Hinde method in this section, including its successes and limitations.

5.1.1 Results of Hinde Method

In this section we will show the genotyping assignments of the Hinde method, and also showcase the effect some of the imposed restrictions have on the genotyping.

Figure 5.2 depicts the genotyping made by the Hinde method of the SNP006 marker. Each color represents a different cluster and thus a different genotype. The assignments are made based on the posterior probability, with the observation being assigned to the cluster to which that observation has the highest posterior probability of belonging to. It is possible to make a stricter assignment, only assigning each observation that has a posterior probability of 0.5 or higher (or other arbitrary threshold). The angles between the lines give us an indication of ploïdy level and the expected ratios between the different ploïdy levels.

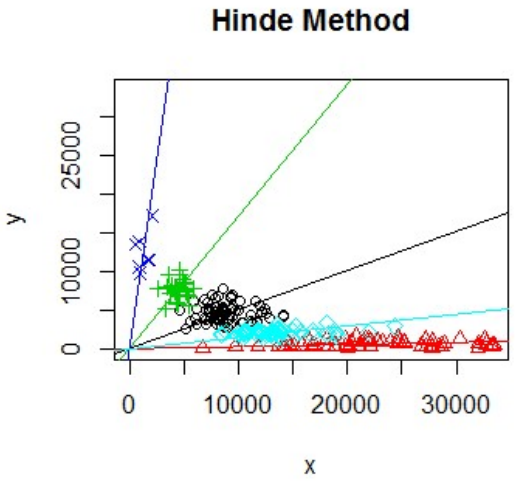


Figure 5.2. The converged genotype assignment made by the Hinde method of the SNP006 marker. Each different color represents another cluster.

Next we will illustrate what the effect of the proposed restrictions is. We will start with imposing Hardy-Weinberg Equilibrium on the prior probabilities in the Hinde Method, which is depicted in figure 5.3. The differences are almost negligible when imposing HWE on the Hinde method in this example. This might imply that the data we are using follows HWE already, and we just improve slightly on the genotyping. The main differences lie in the mixture proportions π_j, which now follow the genotype frequencies of HWE.

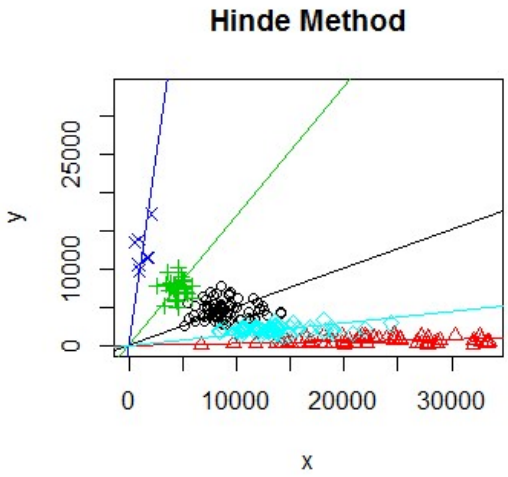


Figure 5.3. The converged genotype assignment made by the Hinde method of the SNP006 marker with HWE imposed.

Figure 5.4 showcases an example where the angles of the regression lines have been set at a predetermined position. In this specific example we made sure the angles where evenly spaced. This restriction is mostly useful if previous data is available and thus the expected angle between clusters is known. Note that the genotype calling now gives quite different results.

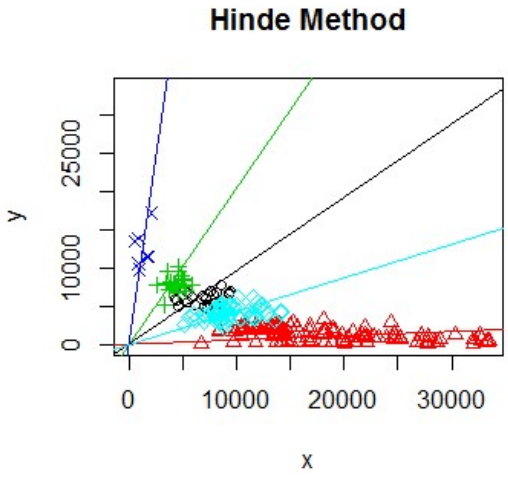


Figure 5.4. The converged genotype assignment made by the Hinde method of the SNP006 marker with evenly spaced angles.

The next example we showcase is where we use the assumed relationship between the allele dosage and signal intensities to calculate the mean of each cluster as described in chapter 3.3.2. The aim is to find the ideal position for the regression lines, based on this relationship. This would be primarily beneficial in the case when not all genotypes are present and we therefore need a way to determine which cluster corresponds to which genotype. For example, with 3 clusters available, it is hard to say if the first cluster corresponds to the simplex or nulliplex case. The results, which are found by constraining the Hinde method to this restriction, can be seen in figure 5.5.

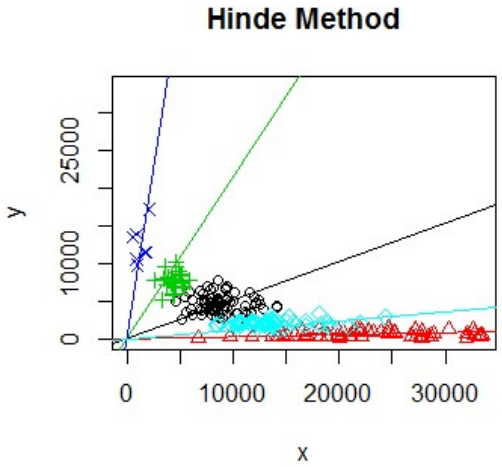


Figure 5.5. The converged genotype assignment made by the Hinde method of the SNP006 marker, in which we assume a linear relationship between signal intensities and allele dosage.

In Figure 5.6 we used multivariate multiple total least squares, which calculates all five regression lines simultaneously as a way of restricting the algorithm. This is one of the differences between total least squares and least squares, as in the case of least squares multivariate multiple least squares gives the same results as fitting the regression lines separately. The method is described in chapter 3.3.2. Theoretically, multivariate multiple total least squares should give a better fit, as all data and weights are taken into account when calculating the regression lines.

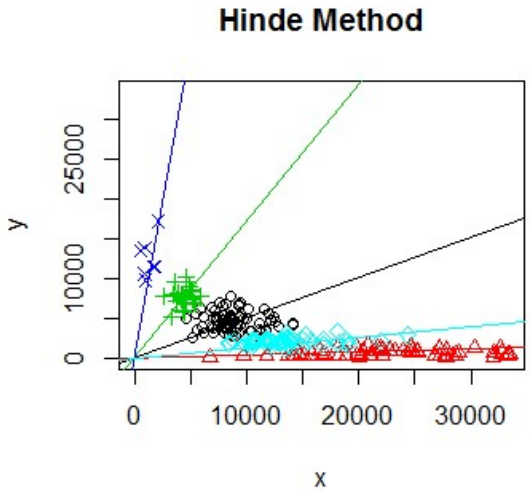


Figure 5.6. The converged genotype assignment made by the Hinde method of the SNP006 marker. We used multiple multivariate total least squares instead of total least squares.

5.1.2 Discussion of Hinde Method

One of the problems we ran into with the Hinde method was the inability of the algorithm to deal with missing clusters, i.e. certain genotypes were missing from the data. In this case, while the algorithm is able to find a set of clusters and assign each observation to a cluster, we have no way of knowing what the ploidy level of each cluster is. In the case of 5 clusters, we can assume the cluster with the lowest y value to be the simplex cluster and the cluster with the highest y value to be the quadruplex cluster. In the case less than 5 clusters are available we are not able to determine this. We can use the BIC criterion to determine if the number of clusters is less than 5, but that would give us no further indication of which allele dosage belongs to which cluster.

FitTetra tries to solve this problem by assuming a relation between signal intensity and allele dosage. This allows prediction of a set of cluster means, which one can use to determine allele dosage of the clusters.

We also tried to impose similar restrictions on the Hinde Method. However, while we are able to predict a similar set of cluster centers by rewriting the assumed relation proposed in the FitTetra paper[1], we are only able to calculate a predicted mean for one variable at a time (I.e. we calculate a mean for the x variable and we calculate a mean for the y variable and combine them to give us a predicted mean). Also, this does not take advantage of the regression method proposed by Hinde, as we are now back to the simpler fitTetra case, in which the second dimension does not influence the assignment, i.e. we are essentially predicting the clusters based on the ratio, as FitTetra does.

The other options we tried were also not satisfying. Restricting the model using a predetermined set of the angles of the regression lines assumes that we have knowledge on the position of the allele dosages beforehand and multivariate multiple least squares does

nothing to help us determine ploidy level. It only calculates all regression lines simultaneously. The only way was to find a way to restrict the regression coefficients or use them in some way to determine ploidy level, but we stranded there, unable to find a solution. We decided that a transformation might help in determining ploidy level and after trying out some different transformations, we came back to the original fitTetra scale, which is the arcsine root of the ratio. This is when we started working on the Conditional Method.

5.2 Conditional Method: Results & Discussion

In this section we will discuss the Conditional method. As stated in the implementation of the conditional method in chapter 4 and in the discussion of the Hinde method in chapter 5, this method aims to stay close to fitTetra by using the same scale (i.e. using the arc sine root transformed ratio of signal intensities), while still employing an extra dimension of information. As stated in chapter 5 there is no golden standard on assessing genotyping assignments, therefore we will mainly focus on visual analysis.

5.2.1 Results of Conditional Method

In this section we will showcase the genotyping assignments made by the conditional method, utilizing mostly a visual approach. We will show the clustering and the assignment of the observations.

Figure 6.2 depicts the genotyping made by the Conditional method of the SNP006 marker. In this example we restrict the μ by imposing a relationship between allele dosage and signal intensity. We also impose equal within cluster variance. This is done because the algorithm converges to a local maxima if we do not impose these restrictions. The assignments are made based on the posterior probability, with the observation being assigned to the cluster to which that observation has the highest posterior probability of belonging to. It is possible to make a stricter assignment, only assigning each observation that has a posterior probability of 0.5 or higher (or another arbitrary threshold). As with fitTetra, the ratio r gives us an indication of allele dosage. The added benefit is the added dimension d. As you can see, the ratio's between the different clusters change across the dimension d, indicating it might give us better genotyping results.

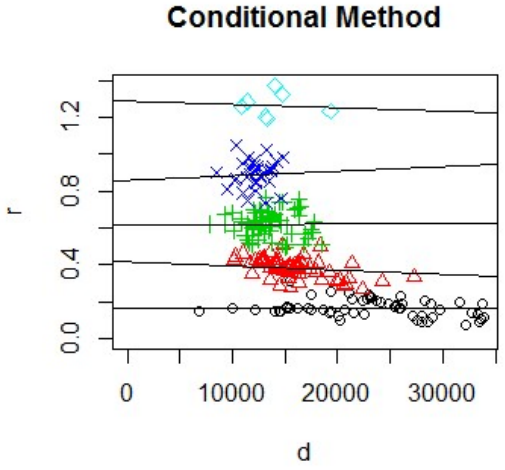


Figure 6.2. The converged genotype assignment made by the conditional method of the SNP006 marker. Restrictions on the variance and means were used.

5.2.2 Discussion of Conditional Method

With this approach we aimed to come close to the original fitTetra method for genotype calling oftetraploid species, while using a bivariate modelling approach with the goal of obtaining better genotyping assignments.

While in essence we succeeded in creating such a method, there are still some road blocks to be overcome. One of the benefits of the original fitTetra was the imposed relation on the μ, where one could impose a linear (or polynomial) relationship between the signal intensities and allele dosages. This allowed the algorithm to predict a set of $\hat{\mu}$ even if not all genotypes were present. In the case of less than 5 clusters, this gives an indication of the number of the clusters and thus gives the user the option of genotyping assignment in the case of less than 5 clusters.

We tried to recreate this method for the conditional method as well. Instead of fitting the intercepts of the regression lines, we forced them to go through the estimated means $\hat{\mu}$. However, the ratio's between the predicted $\hat{\mu}$ are not maintained across the second dimension d. This means that we predicted a ratio of ploidy level, but the regression lines retained to much freedom, meaning the information contained in the predicted means $\hat{\mu}$ lost value the farther away (in the second dimension d) predictions were made for observations. Essentially, observations classified near these predicted means would be restricted by the previously mentioned relationship between the signal intensities and allele dosages, but the farther an observation was from these predicted means, the less this constrained affected them.

Therefore we would like an option to maintain this predicted ratio across the whole of the second dimension d. This would allow us to assign genotypes when not all 5 genotypes are present.

5.3 Simulation Study

In this section we will discuss the simulation study we performed to compare the different methods proposed in this thesis to the fitTetra software. We will start with a discussion of the chosen parameters and models used to create our simulated data. We have created several different simulated data-sets to make sure we compare the methods in different situations. Then we will showcase the results obtained by the three different methods and end this section with a discussion on the results. We will compare the models by looking at the number of correctly genotyped observations compared to the total number of observations.

5.3.1 Parameters and models for simulation study

As discussed in chapter 2.6 we have several parameters we need to choose for our simulation study. The details of how we calculated the simulated data set is also found in chapter 2. Table 5.1 shows our eventual choices of models and parameters for different sets of simulated data. We explain our choices for the different parameters and models below. All data sets contain a sample size of a 1000 observations ($N = 1000$). The values are chosen such that they reflect the datasets we used.

	π_j	μ_j	Σ_j
Set 1	HWE proportions	Linear Model	$\lambda_1 = 10000$, $\lambda_2 = 10000$, $\varphi = \frac{\pi}{4}$
Set 2	HWE proportions	Linear Model	$\lambda_1 = 20000$, $\lambda_2 = 20000$, $\varphi = \frac{\pi}{4}$
Set 3	HWE proportions	Linear Model	$\lambda_1 = 30000$, $\lambda_2 = 30000$, $\varphi = \frac{\pi}{4}$
Set 4	HWE proportions	Quadratic Model	$\lambda_1 = 10000$, $\lambda_2 = 10000$, $\varphi = \frac{\pi}{4}$

Table 5.1. Parameters for the different sets of simulated data. The size of all sets is $N = 1000$.

For all data set, we assume Hardy-Weinberg Equilibrium and calculate our mixture proportions π_j based on that assumption. This gives us the following mixture proportions: $\pi_1 = 0.0625$, $\pi_2 = 0.2500$, $\pi_3 = 0.3750$, $\pi_4 = 0.2500$, and $\pi_5 = 0.0625$.

We used two different models to model the means μ_j. Both models are based on the relationship between fluorescence and allele dosage. In this case we first used a model that assumes a linear relationship and for the second model we assumed a quadratic relationship. In this case values for the fluorescence and background intensities used in these models where chosen by hand. The parameters we choose for the background intensities are $a_0 = 100$, and $b_0 = 150$ and the parameters we choose for the fluorescence intensities are $a_1 = 500$, and $b_1 = 550$. The set of means $\hat{\mu}_j$ for each model is displayed in table 5.3. $\hat{\mu}_j$ are of the general form $\hat{\mu}_j = \begin{pmatrix} \mu_x \\ \mu_y \end{pmatrix}$

Model	$\hat{\mu}_1$	$\hat{\mu}_2$	$\hat{\mu}_3$	$\hat{\mu}_4$	$\hat{\mu}_5$
Linear	$\begin{pmatrix} 100 \\ 2350 \end{pmatrix}$	$\begin{pmatrix} 600 \\ 1800 \end{pmatrix}$	$\begin{pmatrix} 1100 \\ 1250 \end{pmatrix}$	$\begin{pmatrix} 1600 \\ 700 \end{pmatrix}$	$\begin{pmatrix} 2100 \\ 150 \end{pmatrix}$
Quadratic	$\begin{pmatrix} 100 \\ 8950 \end{pmatrix}$	$\begin{pmatrix} 600 \\ 5100 \end{pmatrix}$	$\begin{pmatrix} 2100 \\ 2350 \end{pmatrix}$	$\begin{pmatrix} 4600 \\ 700 \end{pmatrix}$	$\begin{pmatrix} 8100 \\ 150 \end{pmatrix}$

Table 5.3. Means $\hat{\mu}_j$ calculated imposing a linear and quadratic relationship between fluorescence and allele dosage.

For the covariance matrices Σ_j we first use the assumption that all variances are equal, i.e. $\Sigma_j = \Sigma_1 = \Sigma_2 = \Sigma_3 = \Sigma_4 = \Sigma_5$. As stated in chapter 2, we use the eigenvalues to calculate the covariance matrices. Table 5.4 displays the actual covariance matrices for the 4 sets of simulation data. The matrices are of the general form $\Sigma_j = \begin{pmatrix} \sigma_x^2 & \sigma_y\sigma_x \\ \sigma_x\sigma_y & \sigma_y^2 \end{pmatrix}$.

	Covariance
Set 1	$\begin{pmatrix} 10000 & 0 \\ 0 & 10000 \end{pmatrix}$
Set 2	$\begin{pmatrix} 20000 & 0 \\ 0 & 20000 \end{pmatrix}$
Set 3	$\begin{pmatrix} 30000 & 0 \\ 0 & 30000 \end{pmatrix}$
Set 4	$\begin{pmatrix} 10000 & 0 \\ 0 & 10000 \end{pmatrix}$

Table 5.4. Covariance matrices for the 4 sets of data of the simulation study.

As an example of the data generated, figure 5.3 displays the density of the 1th set. The density estimate is found using kernel density estimates. Note how the proportions of each mixture component (which can be identified by the peaks) follow HWE.

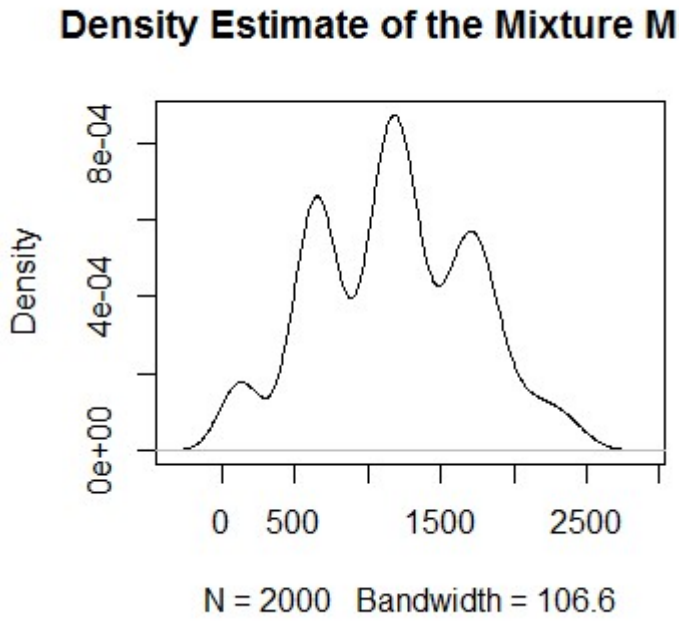


Figure 5.3. Kernel density estimate of the simulated data.

Figure 5.4 displays the data, plotted in two dimensions. The colors represent the different mixture components as defined by the simulation.

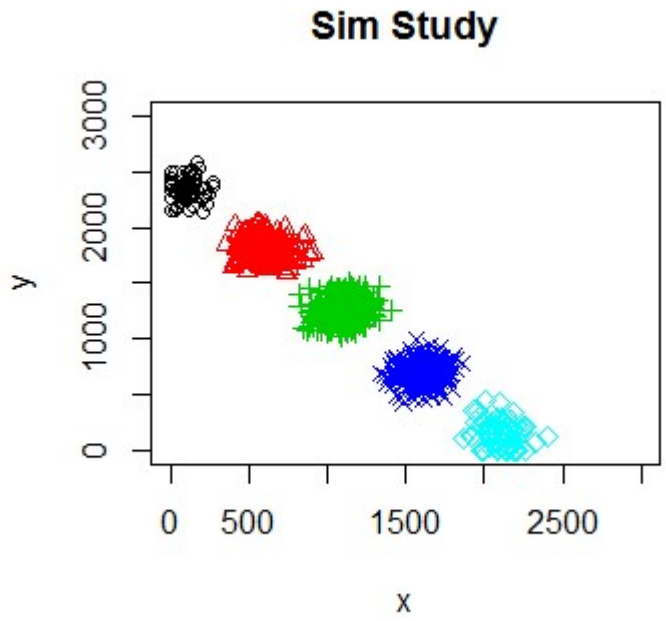


Figure 5.4. Two dimensional plot of the data simulated using the parameters of the 1st set.

5.3.2 Results of the Simulation Study

In this section we showcase the results of the simulation study. The aim of this simulation study is to compare the different methods in a quantative manner and hopefully see some improvement in genotyping for the methods that add an extra dimension. In order to compare the methods in a quantative manner we compare the percentage of successfully genotyped observations over the total amount of observations. Table 5.5 showcases these results.

	Set 1	Set 2	Set 3	Set 4
FitTetra	99.6%	97%	93%	99.1%
Hinde Method	99.9%	97.8%	23.1%	94.7%
Conditional Method	99.8%	98.4%	14%	95.9%

Table 5.5. In this table the obtained percentage of successful genotyped observation is showcased.

As each model also can be constrained in a way it closely resembles the simulated data, by applying Hardy-Weinberg Equilibrium and assuming a relationship between fluorescence intensities and allele dosage, we will use those constraints in obtaining the results.

A visual side by side comparison of the different models is given in figures 5.5-5.7. This is mostly done to show how the different models fit the results, however, the only

reasonable comparison that can be made between the different models is by comparing the number of accurately determined genotypes.

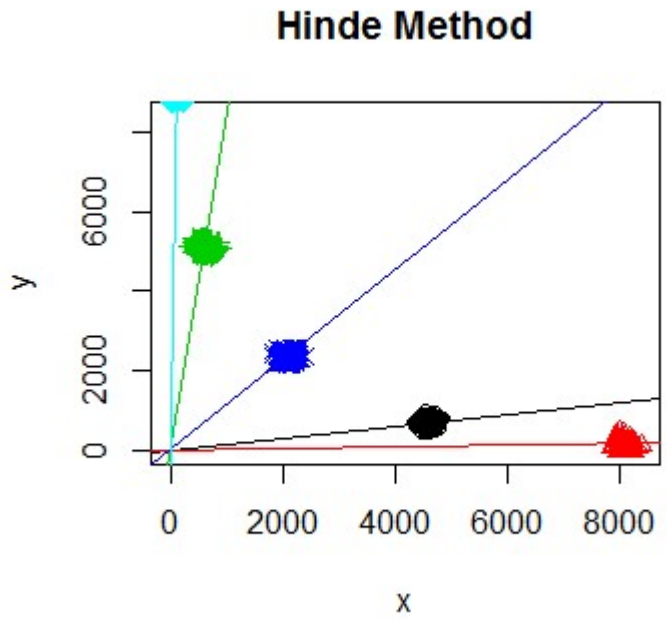


Figure 5.5. Hinde method fit on the 1st set of simulated data.

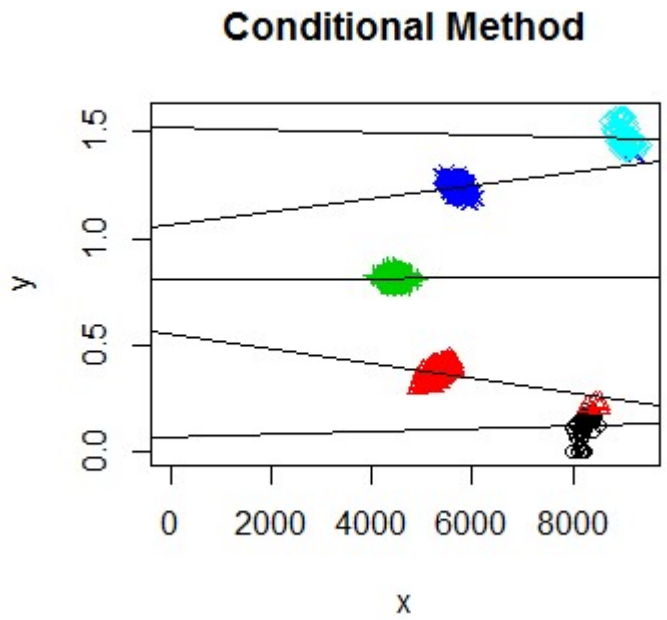


Figure 5.6. Conditional method fit on the 1st set of simulated data.

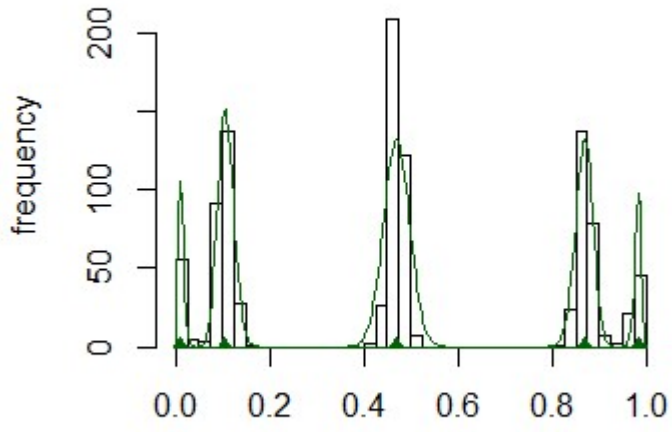


Figure 5.7. fitTetra fit on the 1st set of simulated data.

5.3.3 Discussion of the Simulation Study

With the results we obtained we can try to make a comparison between the different methods. What is interesting to see is that the Hinde method and the Conditional method both outperform fitTetra in the first two sets, although very slightly, in the case were we used a linear model and lower variance to generate the simulation study.

However, it seems that once the variance hit a certain threshold the performance of both the Hinde method and Conditional method seems to drop off steeply. Both methods use linear regression to determine the posterior probabilities. However, no further constraints were placed on these linear regression lines, such as the constraints placed on the means. This means the regression lines gain too much freedom to move and the constraints we place on the means are not enough. To further develop these methods, we would need a way to place constraints on the regression lines in a similar way to the way we constrain the means, i.e. placing a constraint based on the assumption that there is a relationship between fluorescence and allele dosage.

Reference List

1. Hsu, TM; Chen, X; Duan, S; Miller, RD; Kwok, PY: **Universal SNP genotyping assay with fluorescence polarization detection**. Biotechniques. 2001, 31(3):560, 562, 564-8, passim.
2. Markwith, S.H.; Stewart, D.J.; Dyer, J.L.: **TETRASAT: a program for the population analysis of allotetraploid microsatellite data.** Molecular Ecology Notes 2006, 6, 586-589
3. http://www.illumina.com/techniques/microarrays/array-data-analysis-experimental-design/genomestudio.html
4. Voorrips, Gort; Vosman, Ben: **Genotype calling in tetraploid species from bi-allelic marker data using mixture models.** Bioinformatics 2011, 12:172
5. Hinde, John; Coffey, Norma; Garcia, Franco: **Finite Mixture Model Clustering of SNP Data.** Channel Network Conference 3rd July 2013
6. Rippe, Ralph C.A.; Meulman, Jacqueline J.; Eilers, Paul H.C.: **Reliable Single Chip Genotyping with Semi-Parametric Log-Concave Mixtures.** PLoS ONE 7(10): e46267.doi:10.1371/journal.pone.0046267
7. McLachlan, Geoffrey; Peel, David: **Finite Mixture Models.** John Wiley 2000
8. Markovsky, Ivan; Van Huffel, Sabine. **Overview of total least squares methods. Signal Process.** 2007, 87:2283 – 2302.
9. Fraley, Chris; Raftery, Adrian E.: **Model-based Clustering.** Discriminant Analysis and Density Estimation Journal of the American Statistical Association 2002, 97:611-631
10. Anithakumari, AM; Tang, J; van Eck, HJ; Visser, RFG; Leunissen, JAM; Vosman, B; van der Linden, CG: **A pipeline for high throughput detection and mapping of SNPs from EST databases.** Mol. Breeding 2010, 26: 65-75
11. Hathaway, Richard J. **A Constrained Formulation of Maximum-Likelihood Estimation for Normal Mixture Distributions.** Ann. Statist. 13 1985, no. 2, 795--800.
12. Chen, Jiahua; Tan, Xianming: **Inference for multivariate normal mixtures.** Journal of Multivariate Analysis, Volume 100, Issue 7, 2009, 1367-1383

Appendix A: Hinde Method Code

```
#Hinde Method for Finite Mixture Model Clustering of SNP data

##############################################################################

#Clustering Algorithm

#

#Parameters:

#xh: raw intensity of the first channel

#yh: raw intensity of the second channel

#ng: total number of possible clusters (one more than ploidy level)

# return value: list with 4 elements:

# $clus.mu: matrix of with ng rows and 2 columns with cluster means

# $clus.sd: list containing the ng number of covariance matrices, one for each clus
ter

# $clus.p: proportions of the clusters, assuming 5 clusters.

# $cluster: assuming 5 clusters, which observation are in which cluster.


ClusterInitKmeans <- function(xh, yh, ng = 5) {

  require(mvtnorm)

  x <- cbind(xh, yh)


  # Try 1 to ng clusters, decide on number of clusters, fill in gaps

  km <- list()

  wss <- list()

  BIC <- numeric(ng)


  for (i in 1:ng){

    km[[i]] <- kmeans(x, centers=i, nstart=10*i)   # try different starts, increasin
g numbers for more clusters

    wss[[i]] <- km[[i]]$withinss
```

```
    cluster <- list()

    clus.sd <- list()


    clus.mu <- km[[i]]$centers              #i by 2 matrix of cluster centers

    for(j in 1:nrow(clus.mu)){ #Creating covariance matrices for each cluster

      cluster[[j]] <- cbind(x[km[[i]]$cluster == j, 1], x[km[[i]]$cluster == j, 2])
#defining the clusters

      clus.sd[[j]] <- cov.wt(cluster[[j]], center = clus.mu[j,]) #covariance matrix
, using the found mu's as center

    }

    clus.p <- km[[i]]$size/nrow(x)       #vector of fraction of observations in each
cluster


    #Calculating log likelihood and BIC

    if (i==1) loglik <- sum(log(dmvnorm(cluster[[j]], clus.mu, clus.sd[[j]]$cov)))

    else {

    clusterloglik <- numeric(nrow(clus.mu))

    for(k in 1:nrow(clus.mu)){

      clusterloglik[[k]] <- sum(log(dmvnorm(cluster[[k]], clus.mu[k,], clus.sd[[k]]
$cov)))

    }

    loglik <- sum(clusterloglik)

    }


    BIC[i] <- -2*loglik + log(nrow(x))*(7*i)    # use more heavy penalty than ordina
ry BIC to avoid spurious clusters

  }


  nrclust <- which.min(BIC)


  extraclust <- ng-nrclust
```

```
  clus.mu <- km[[nrclust]]$centers #ng by 2 matrix

  eucl.dist.mu <- sqrt((clus.mu[,1]^2)+(clus.mu[,2]^2)) #used to order the mu's

  o <- order(eucl.dist.mu)


  seq2 <- function(low,up,n) seq(low,up, length.out=n+2)[-c(1,n+2)]  # help functio
n for filling gaps


  if (extraclust > 0) {


    # Fill the gaps, assuming that chosen mu's so far are consecutive;

    # so, fill only on the left side and/or on the right side.


    lo.mu <- clus.mu[o[1],]

    min.at <- o[1]

    lo.cluster <- cbind(x[km[[nrclust]]$cluster==min.at, 1], x[km[[nrclust]]$cluste
r==min.at, 2])

    eucl.dist.lo <- sqrt((lo.cluster[,1]^2)+(lo.cluster[,2]^2))

    lo.x <- x[which.min(eucl.dist.lo),]


    hi.mu <- clus.mu[o[nrclust],]

    max.at <- o[nrclust]

    hi.cluster <- cbind(x[km[[nrclust]]$cluster==max.at, 1], x[km[[nrclust]]$cluste
r==max.at, 2])

    eucl.dist.hi <- sqrt((hi.cluster[,1]^2)+(hi.cluster[,2]^2))

    hi.x <- x[which.max(eucl.dist.hi),]


    lo.empty <- lo.x

    lo.eucl <- min(eucl.dist.lo)

    hi.empty <- hi.x

    hi.eucl <- max(eucl.dist.hi)
```

```
  if (lo.eucl == 0) {n.lo <- 0; n.hi <- extraclust}
  else if (hi.eucl == 0) {n.lo <- extraclust; n.hi <- 0}
  else {
    n.lo <- 0:extraclust
    n.hi <- extraclust-n.lo
    divide <- (lo.empty/(n.lo+1))/(hi.empty/(n.hi+1))
    x <- which.min(abs(log(divide)))
    n.lo <- n.lo[x]; n.hi <- n.hi[x]
  }
  clus.mu <- c(seq2(0, lo.mu, n.lo), clus.mu, seq2(hi.mu, maxy, n.hi))
}


  clus.sd <- list()
  cluster <- list()
  for(j in 1:nrclust){ #Creating covariance matrices for each cluster
    cluster[[j]] <- cbind(x[km[[i]]$cluster == j, 1], x[km[[i]]$cluster == j, 2]) #
defining the clusters
    clus.sd[[j]] <- cov.wt(cluster[[j]], center = clus.mu[j,])$cov #covariance matr
ix, using the found mu's as center
  }


  clus.p <- km[[5]]$size/nrow(x)


list(clus.mu=clus.mu, clus.sd=clus.sd, clus.p = clus.p, cluster = cluster)
}
#Clustering Algorithm based on Mclust
#
#Parameters:
#xh: raw intensity of the first channel
#yh: raw intensity of the second channel
#ng: total number of possible clusters (one more than ploidy level)
```

```r
# return value: list with 4 elements:

# $clus.mu: matrix of with ng rows and 2 columns with cluster means

# $clus.sd: list containing the ng number of covariance matrices, one for each clus
ter

# $clus.p: proportions of the clusters, assuming 5 clusters.

# $cluster: assuming 5 clusters, which observation are in which cluster.


ClusterInitMclust <- function(xh, yh, ng = 5){

  require(mclust)


  x <- cbind(xh,yh)


  mod <- Mclust(x, G = ng)


  mu <- t(mod$parameters$mean)

  var <- lapply(seq(dim(mod$parameters$variance$sigma)[3]), function(x) mod$paramet
ers$variance$sigma[ , , x])

  p <- mod$parameters$pro

  ind <- mod$classification


  cluster <- list()


  for(j in 1:ng){

    cluster[[j]] <- cbind(x[ind == j, 1], x[ind == j, 2]) #defining the clusters

  }




  list(clus.mu = mu, clus.sd = var, clus.p = p, cluster = cluster)

}


#############################################################################
#######
```

```r
#Regression through clusters


#Total Least Squares Regression
#Helper function to find the beta's in total least squares through SVD decompositio
n
#
#xdata: indepent variable's
#ydata: dependent variable
#return value is beta


tlsRegression <- function(xdata, ydata){
  m <- length(ydata)
  X <- as.matrix(xdata)
  Y <- as.matrix(ydata)
  n <- ncol(X)
  Z <- cbind(X,Y)
  SVDZ <- svd(Z)
  VXY <- SVDZ$v[1:n,(1+n):ncol(SVDZ$v)]
  VYY <- SVDZ$v[(1+n):ncol(SVDZ$v),(1+n):ncol(SVDZ$v)]
  B <- -VXY/VYY
}
#Weighted Variance
#
#xdata: indepent variable's
#ydata: dependent variable's
#x.hat: list of x.hat obtained from reflection on orthogonal regression line
#y.hat: list of y.hat obtained from reflection on orthogonal regression line
#Z: matrix of weights. Based on the likelihood of an observation to be in a cluster
weightedVar <- function(xdata, ydata, x.hat, y.hat, Z, eqvar = F){
  Z <- as.matrix(Z)
  r <- ncol(Z)
```

```
  var <- numeric(ncol(Z))
  wt <- apply(Z, 2, function(x) x/sum(x))
  x.hat2 <- unlist(lapply(1:r, function(x) x.hat[[x]]))
  y.hat2 <- unlist(lapply(1:r, function(x) y.hat[[x]]))

  for(i in 1:r){
    varx <- sum(wt[, i]*(xdata - x.hat[[i]])^2)
    vary <- sum(wt[, i]*(ydata - y.hat[[i]])^2)
    var[i] <- (varx + vary)/2
  }
  if(eqvar == T){
    varx <- sum(as.vector(wt)*(rep(xdata, times = ncol(Z)) - x.hat2)^2)
    vary <- sum(as.vector(wt)*(rep(ydata, times = ncol(Z)) - y.hat2)^2)
    var <- (varx + vary)/2
    var <- rep(var, times = ncol(Z))
  }

  return(var)
}
#Weighted Total Least Squares
#
#xdata: indepent variable's
#ydata: dependent variable's
#Z: matrix of weights. Based on the likelihood of an observation to be in a cluster
#returns a list with 3 elements:
#x.hat: a matrix or vector with x.hat values
#y.hat: a vector with y.hat values
#std.err: standard error assuming equal variance in x and y
weightedTLS <- function(xdata, ydata, Z){
  r <- ncol(Z)
  n <- length(ydata)
```

```
  results <- list()
  x <- cbind(xdata,ydata)
  Dm <- lapply(1:r, function(i) {diag(sqrt(Z[,i]))%*%x%*%diag(1, nrow = 2) })

  beta <- sapply(1:r, function(i){tlsRegression(Dm[[i]][,1], Dm[[i]][,2])})
  x.hat <- lapply(1:r, function(i) (ydata*beta[i] + xdata)/(1 + beta[i]^2))
  y.hat <- lapply(1:r, function(i) x.hat[[i]]*beta[i])
  var <- weightedVar(xdata, ydata, x.hat, y.hat, Z)

  results <- list(x.hat = x.hat, y.hat = y.hat, var = var, beta = beta)

  return(results)
}
#Initial Beta
#
#xdata: indepent variable's
#ydata: dependent variable's
#cluster: the list of clusters obtained from the initial clusters

BetaInit <- function(xdata, ydata, clusters){
  beta <- numeric(length(clusters))
  x.hat <- list()
  y.hat <- list()

  mu <- list()
  var <- numeric()

  for(i in 1:length(clusters)){
    beta[i] <- tlsRegression(clusters[[i]][,1],clusters[[i]][,2])
    n <- length(clusters[[i]][,1])
```

```
    x.hat.tmp <- (clusters[[i]][,2]*beta[i] + clusters[[i]][,1])/(1 + beta[i]^2)
    y.hat.tmp<- x.hat.tmp*beta[i]
    x.hat[[i]] <- (ydata*beta[i] + xdata)/(1 + beta[i]^2)
    y.hat[[i]] <- x.hat[[i]]*beta[i]
    var[i] <- (sum((clusters[[i]][,1] - x.hat.tmp)^2) + sum((clusters[[i]][,2] - y.
hat.tmp)^2))/(2*(n - 1))
    mu[[i]] <- cbind(x.hat[[i]], y.hat[[i]])
  }

  var <- lapply(var, function(x) diag(x, nrow = 2))

  return(list(mu = mu, beta = beta, x.hat = x.hat, y.hat = y.hat, var = var))
}
##############################################################################
####
#EM algorithm

#Log Likelihood
#
#xh: raw intensity of the first channel
#yh: raw intensity of the second channel
#p: proportions of the mixtures
#mu: centers of the mixtures, supplied as a matrix with each row representing a mix
ture
#sigma: std. err of the mixtures, supplied as a list of matrices, each matrix repre
sinting a mixture

loglikf <- function(xh, yh, p, mu, sigma){
  require(mvtnorm)
  #package to calculate multivariate normal distribution

  x <- cbind(xh,yh)
```

```
  densities <- matrix(0, ncol = length(sigma), nrow = length(xh))
  for(i in 1:length(sigma)){
    for(j in 1:length(xh)){
      densities[j, i] <- dmvnorm(x[j,], mu[[i]][j,], sigma[[i]])
    }
  }

  D <- densities
  #multivariate normal densities, dimensions n x ng.
  DP <- D*p
  #densities * prior probabilities
  rSDP <- rowSums(DP) #likelihood of observation
  LL <- sum(log(rSDP)) #complete log likelihood

  list(LL=LL, D=DP, sumD=rSDP)
}

#Posterior probabilities for each observation
#
#D: matrix of prior probabilities * densities (pi*fi)
#sumD: vector of pi*fi/sum(pi*fi)

EMGaussExp <- function(D, sumD){

  #Ti's almost calculated in the loglikihood function, just need to divide pi*fi/su
m(pi*fi)
  Z <- t(scale(t(D), center = F, scale = sumD)) #double transpose because scale onl
y affects columns
  attributes(Z)$"scaled:scale" <- NULL
  Z #Z is a matrix with posterior probabilities
```

```
}

#Assign clusters based on posterior probabilities
#
#xh: raw intensity of the first channel
#yh: raw intensity of the second channel
#Z: a matrix with posterior probabilities for each observation

EMCluster <- function(xh, yh, Z){
  ind <- apply(Z, 1, which.max) #indicator matrix with cluster

  cluster <- lapply(1:ncol(Z), function(i) data.frame(x = xh[ind == i], y = yh[ind
== i]))

  cluster
}
#Proportions based on HWE
#
#Z: a matrix with posterior probabilities for each observation

EMMax.p.HW <- function(Z) {
  p <- apply(Z,2,mean)
  phw <- (4*p[1] + 3*p[2] + 2*p[3] + 1*p[4] + 0*p[5])/4
  c(phw^4,4*phw^3*(1-phw),6*phw^2*(1-phw)^2,4*phw*(1-phw)^3,(1-phw)^4)
}

#Find new proportions
#
#Z: a matrix with posterior probabilities for each observation
#ptype: 'p.free' for free estimation of proportions, 'p.HW' for HWE proportions
```

```
EMMaxP <- function(Z, ptype = 'p.free'){
  p <- switch(ptype,
              p.free = apply(Z,2,mean),
              p.HW = EMMax.p.HW(Z))
  p
}

#Finding Mu
#
#xh: raw intensity of the first channel
#yh: raw intensity of the second channel
#beta: a vector with beta's obtained from EMMaxBeta
#mutype: See notes

EMMaxMu <- function(xh, yh, beta){
  mu <- list()
  r <- length(beta)

  x.hat <- lapply(1:r, function(i) (yh*beta[i] + xh)/(1 + beta[i]^2))
  y.hat <- lapply(1:r, function(i) x.hat[[i]]*beta[i])
  mu <- lapply(1:r, function(i) cbind(x.hat[[i]], y.hat[[i]]))

  return(list(mu = mu, x.hat = x.hat, y.hat = y.hat))
}

#Restrictions for variance
#
#xh: raw intensity of the first channel
#yh: raw intensity of the second channel
#Z: a matrix with posterior probabilities for each observation
```

```
EMMaxVar <- function(xh, yh, Z, x.hat, y.hat, vartype = 0, sd = c(0,0,0,0,0)){

  r <- ncol(Z)

  x.hat2 <- unlist(lapply(1:r, function(x) x.hat[[x]]))
  y.hat2 <- unlist(lapply(1:r, function(x) y.hat[[x]]))
  n <- length(x.hat2)

  if(vartype == 0){
    var <- weightedVar(xh, yh, x.hat, y.hat, Z)
  }

  else{
    switch(vartype,
           var <- weightedVar(xh, yh, x.hat2, y.hat2, Z, eqvar = T),
           var <- sd^2)
  }

  var <- lapply(var, function(x) diag(x, nrow = 2))

  return(var)
}
#Max Beta
#
#xh: raw intensity of the first channel
#yh: raw intensity of the second channel
#Z: a matrix with posterior probabilities for each observation
EMMaxBeta <- function(xh, yh, Z, betatype = 0){

  r <- ncol(Z)
```

```
  results <- weightedTLS(xh, yh, Z)

  beta <- sapply(1:r, function(x) results$beta[[x]])
  o <- order(beta, decreasing = T)

  angles <- atan(beta) * 180 / pi
  evenly <- seq(min(angles),max(angles), length.out = 5)

  Dmx <- sapply(1:r, function(i) {diag(sqrt(Z[,i]))%*%xh})
  Dmy <- sapply(1:r, function(i) {diag(sqrt(Z[,i]))%*%yh})

  switch(betatype,
         beta <- orderise(tan(evenly*(pi/180)),o),
         beta <- restrictBeta(xh, yh, Z, beta = beta),
         beta <- diag(tlsRegression(Dmx, Dmy)))

  return(beta)
}
#Helper function to get the correct order of beta
orderise <- function(vector, order){
  n <- length(vector)
  o <- order
  beta <- numeric(n)
  for(i in 1:n){
    beta[o[i]] <- sort(vector, partial = n+1-i)[n+1-i]
  }
  return(beta)
}
#Beta Restrictions, calculating specific restrictions for beta
restrictBeta <- function(xh, yh, Z, beta){
```

```
  vector <- beta

  r <- ncol(Z)

  o <- order(vector, decreasing = T)


  #calculate centers.

  mu <- restrictMu(xh, yh, Z, order = o)


  beta2 <- numeric(r)


  for(i in 1:r){

    beta2[i] <- mu[i,2]/mu[i,1]

  }


  #New beta's

  beta <- orderise(beta2, order = o)


  return(beta)

}

#Restricting cluster centers

restrictMu <- function(xh, yh, Z, order){


  ng <- ncol(Z)

  nr <- nrow(Z)


  #To make sure the correct genotypes are assesed, ordering Z from highest to lowes
t)

  Z <- Z[, order]


  xi <- rep(xh, ng)

  yi <- rep(yh, ng)

  wgt <- as.vector(Z)
```

```
  #Highest beta = Highest genotype
  x <-rep((ng - 1):0, each = nr)
  x2 <- rep((ng - 1):0, each = nr)

  formulx <- log(xi) ~ x
  formuly <- log(yi) ~ x2

  xmod <- lm(formulx, weights = wgt)
  ymod <- lm(formuly, weights = wgt)

  mux <- predict(xmod, data.frame(x=0:(ng-1)),type="response")
  muy <- predict(ymod, data.frame(x2=0:(ng-1)),type="response")

  mu <- cbind(mux, muy)
  mu <- exp(mu)

  return(mu)

}
#Calculating angles based on multivariate multiple least squares
fixAngle <- function(xh, yh, Z){
  r <- ncol(Z)

  Dmx <- sapply(1:r, function(i) {diag(sqrt(Z[,i]))%*%xh})
  Dmy <- sapply(1:r, function(i) {diag(sqrt(Z[,i]))%*%yh})

  beta <- diag(tlsRegression(Dmx, Dmy))

  angles <- atan(beta) * 180 / pi
```

```
}
#Maximisation function
#
#xh: raw intensity of the first channel
#yh: raw intensity of the second channel
#Z: a matrix with posterior probabilities for each observation

EMMax <- function(xh, yh, Z, betatype = 0, vartype = 0, ptype = 'p.free', sd = c(0,
0,0,0,0)){
  p <- EMMaxP(Z, ptype) #finds max proportions
  beta <- EMMaxBeta(xh, yh, Z, betatype)
  res <- EMMaxMu(xh, yh, beta)
  mu <- res$mu
  x.hat <- res$x.hat
  y.hat <- res$y.hat
  var <- EMMaxVar(xh, yh, Z, x.hat, y.hat, vartype, sd)



  list(p = p, mu = mu, var = var, beta = beta)
}


#EM Algorithm iterative
#
#xh: raw intensity of the first channel
#yh: raw intensity of the second channel
#ng: total number of possible clusters (one more than ploidy level)
#p.start: starting values for p
#mu.start: starting values for mu
#sd.start: starting values for the standard deviations
```

```
EMMix <- function(xh, yh, ng = 5, p.start, mu.start, sd.start, maxiter = 50, betaty
pe = 0, vartype = 0,
                  ptype = 'p.free', sd = c(0,0,0,0,0)){

  LLnw <- loglikf(xh, yh, p.start, mu.start, sd.start)
  iter <- 0




    repeat { # start of EM loop

      iter <- iter+1
      LL <- LLnw

      Z <- EMGaussExp(LL$D, LL$sumD)
      cluster <- EMCluster(xh, yh, Z)
      psi <- EMMax(xh, yh, Z, betatype, vartype, ptype, sd)
      LLnw <- loglikf(xh, yh,psi$p, psi$mu, psi$var)

      if (is.na(LLnw$LL) || (max(abs(LLnw$LL - LL$LL)) < 0.0000001)||
            (maxiter>0 && iter>maxiter) ) break
    } # end of EM loop
    list(p = psi$p, mu = psi$mu, var = psi$var, beta = psi$beta, iter= iter, LL = L
L$LL, LLstart = loglikf(xh, yh, p.start, mu.start, sd.start)$LL, Z = Z, cluster = c
luster)
}
##############################################################################
#
#Complete Hinde Method
#
#
```

```
HindeMethod <- function(xh, yh, ng = 5, plotreg = T, maxiter = 0, type = 'Mclust',
betatype = 0, vartype = 0,
                        ptype = 'p.free', sd = c(0,0,0,0,0)){
  clustering <- switch(type,
                       Kmeans = ClusterInitKmeans(xh, yh, ng),
                       Mclust = ClusterInitMclust(xh, yh, ng))
  initBeta <- BetaInit(xh,yh, clustering$cluster)
  results <- EMMix(xh, yh, ng, p.start = clustering$clus.p, mu.start = initBeta$mu,
sd.start = initBeta$var, maxiter = maxiter, betatype = betatype, vartype = vartype,
                   ptype = ptype, sd = sd)

  if(plotreg == T){
    SimplePlot(xh, yh, clusters = results$cluster, beta = results$beta)
  }

  return(results)
}

#Helper function to plot the bivariate normal density
#
#
plotBivariateNormal <- function(clusters, beta, center, sigma, n = 100, xlabel = "X
_Raw", ylabel = "Y_Raw"){

  #sigma.inv <- list()

  #for(i in 1:length(clusters)){
  #sigma.inv[[i]] = solve(sigma[[i]], matrix(c(1,0,0,1),2,2))
  #}
```

```
  #ellipse <- function(s,t, center, sigma.inv) {u<-c(s,t)-center; u %*% sigma.inv %
*% u / 2}


  max.x <- numeric(length(clusters))

  max.y <- numeric(length(clusters))


  for(i in 1:length(clusters)){

  max.x[i] <- max(clusters[[i]][,1])

  max.y[i] <- max(clusters[[i]][,2])

  }


  x <- list()

  y <- list()

  z <- list()


  #x <- (0:(n-1)) * (max(max.x)/(n-1))

  #y <- (0:(n-1)) * (max(max.y)/(n-1))


  for(i in 1:length(clusters)){

  x[[i]] <- (0:(n-1)) * (max.x[i]/(n-1))

  y[[i]] <- (0:(n-1)) * (max.y[i]/(n-1))


  #z[[i]] <- mapply(ellipse, as.vector(rep(x[[i]],n)), as.vector(outer(rep(0,n), y[
[i]], `+`)), center[i,], sigma.inv[[i]])


  z[[i]] <- -log(dmvnorm(cbind(as.vector(rep(x[[i]],n)), as.vector(outer(rep(0,n),
y[[i]], `+`))), mean = center[i,], sigma = sigma[[i]]))

  #z[[i]] <- -log(dmvnorm(cbind(as.vector(rep(x,n)), as.vector(outer(rep(0,n), y, `
+`))), mean = center[i,], sigma = sigma[[i]]))

  }
```

```
  plot(1, type="n", xlim=c(0,max(max.x)), ylim=c(0,max(max.y)), xlab=xlabel, ylab=y
label)


  for(i in 1:length(clusters)){
  points(clusters[[i]], pch=i, col = i)
  abline(a = 0, b = beta[i], col = 'red')
  contour(x[[i]],y[[i]],matrix(z[[i]],n,n), nlevels=5, col = terrain.colors(11), ad
d=TRUE)
  #contour(x,y,matrix(z[[i]],n,n), nlevels=5, col = terrain.colors(11), add=TRUE)
  }
}


SimplePlot <- function(xh, yh, clusters, beta){
  plot(c(0,max(xh)), c(0,max(xh)), type = 'n', xlab = "x", ylab = "y", main = "Hind
e Method")


  for(i in 1:length(clusters)){
    points(clusters[[i]], pch=i, col = i)
    abline(a = 0, b = beta[i], col = i)
  }
}
```

Appendix B: Conditional Method Code

```
#Bivariate Method for Finite Mixture Model Clustering of SNP data


##############################################################################

#Clustering Algorithm based on Mclust

#

#Parameters:

#xh: raw intensity of the first channel

#yh: raw intensity of the second channel

#ng: total number of possible clusters (one more than ploidy level)

# return value: list with 4 elements:

# $clus.mu: matrix of with ng rows and 2 columns with cluster means

# $clus.sd: list containing the ng number of covariance matrices, one for each clus
ter

# $clus.p: proportions of the clusters, assuming 5 clusters.

# $cluster: assuming 5 clusters, which observation are in which cluster.


ClusterInitMclust <- function(x, y, ng = 5){

  require(mclust)


  data <-  cbind(y,x)


  mod <- Mclust(y, G = ng)


  mu <- t(mod$parameters$mean)

  o <- order(mu)

  mu <- mu[o]

  var <- mod$parameters$variance$sigma^2

  p <- mod$parameters$pro
```

```
  p <- p[o]
  ind <- mod$classification

  cluster <- list()

  for(j in 1:ng){
    cluster[[j]] <- cbind(data[ind == j, 1], data[ind == j, 2]) #defining the clust
ers
  }
  cluster <- cluster[o]

  list(mu = mu, var = var, p = p, cluster = cluster)
}
#Clustering Algorithm based on Kmeans
#
#Parameters:
#xh: raw intensity of the first channel
#yh: raw intensity of the second channel
#ng: total number of possible clusters (one more than ploidy level)
# return value: list with 4 elements:
# $clus.mu: matrix of with ng rows and 2 columns with cluster means
# $clus.sd: list containing the ng number of covariance matrices, one for each clus
ter
# $clus.p: proportions of the clusters, assuming 5 clusters.
# $cluster: assuming 5 clusters, which observation are in which cluster.
ClusterInitKmeans <- function(x, y, ng = 5){

  data <-  cbind(y,x)

  km <- kmeans(y, centers=ng, nstart=10*ng)   # try different starts, increasing num
bers for more clusters
  wss <- km$tot.withinss
```

```
  mu <- km$centers              #i*1 matrix of cluster centers (transformed scale)
  o <- order(mu)
  mu <- mu[o]
  var <- (wss/(length(y)-ng)) #one numeric value: overall within-cluster var (trans
formed scale)
  p <- km$size/length(y)
  p <- p[o]
  # Try 1 to ng clusters, decide on number of clusters, fill in gaps
  ind <- km$cluster

  cluster <- list()

  for(j in 1:ng){
    cluster[[j]] <- cbind(data[ind == j, 1], data[ind == j, 2]) #defining the clust
ers
  }
  cluster <- cluster[o]

  return(list(mu=mu, var=var, p = p, cluster = cluster))
}
##############################################################################
#######
#Initial Regression Lines
#
#
BetaInit <- function(x, y, cluster){
  r <- length(cluster)

  #Beta calculated with reflection of mu on the y-axis as intercept using scaled da
ta
```

```
  beta <- sapply(1:r, function(i) lm(cluster[[i]][,1]~ cluster[[i]][,2])$coefficien
ts[2])
```

```
  intercept <- sapply(1:r, function(i) lm(cluster[[i]][,1] ~ cluster[[i]][,2])$coef
ficients[1])


  y.hat <- lapply(1:r, function(i) intercept[i] + x*beta[i])


  y.hat.tmp <- lapply(1:r, function(i) intercept[i] + cluster[[i]][,2]*beta[i])


  n <- sapply(1:r, function(i) length(cluster[[i]][,1]))
  vary <- sapply(1:r, function(i) sum(((cluster[[i]][,1] - y.hat.tmp[[i]])^2)/(n[i]
-1)))


  return(list(intercept = intercept, beta = beta, y.hat = y.hat, vary = vary))
}
##############################################################################
#######
#EM algorithm


#Log Likelihood
#
#xh: raw intensity of the first channel
#yh: raw intensity of the second channel
#p: proportions of the mixtures
#mu: centers of the mixtures, supplied as a matrix with each row representing a mix
ture
#sigma: std. err of the mixtures, supplied as a list of matrices, each matrix repre
sinting a mixture


loglikf <- function(y, p, y.hat, vary){


  densities <- matrix(0, ncol = length(vary), nrow = length(y))
```

```
  for(i in 1:length(vary)){
    for(j in 1:length(y)){
      densities[j, i] <- dnorm(y[j], y.hat[[i]][j], sqrt(vary[i]))
    }
  }


  D <- densities
  #multivariate normal densities, dimensions n x ng.
  DP <- D*p
  #densities * prior probabilities
  rSDP <- rowSums(DP) #likelihood of observation
  LL <- sum(log(rSDP)) #complete log likelihood

  list(LL=LL, D=DP, sumD=rSDP)
}

#Posterior probabilities for each observation
#
#D: matrix of prior probabilities * densities (pi*fi)
#sumD: vector of pi*fi/sum(pi*fi)

EMGaussExp <- function(D, sumD){

  #Ti's almost calculated in the loglikihood function, just need to divide pi*fi/su
m(pi*fi)
  Z <- t(scale(t(D), center = F, scale = sumD)) #double transpose because scale onl
y affects columns
  attributes(Z)$"scaled:scale" <- NULL
  Z #Z is a matrix with posterior probabilities
}
```

```r
#Assign clusters based on posterior probabilities
#
#xh: raw intensity of the first channel
#yh: raw intensity of the second channel
#Z: a matrix with posterior probabilities for each observation

EMCluster <- function(x, y, Z){
  ind <- apply(Z, 1, which.max) #indicator matrix with cluster

  cluster <- lapply(1:ncol(Z), function(i) data.frame(x = x[ind == i], y = y[ind == i]))

  cluster
}
#Proportions based on HWE
#
#Z: a matrix with posterior probabilities for each observation

EMMax.p.HW <- function(Z) {
  p <- apply(Z,2,mean)
  phw <- (4*p[1] + 3*p[2] + 2*p[3] + 1*p[4] + 0*p[5])/4
  c(phw^4,4*phw^3*(1-phw),6*phw^2*(1-phw)^2,4*phw*(1-phw)^3,(1-phw)^4)
}

#Find new proportions
#
#Z: a matrix with posterior probabilities for each observation
#ptype: 'p.free' for free estimation of proportions, 'p.HW' for HWE proportions

EMMaxP <- function(Z, ptype = 'p.free'){
```

```
  p <- switch(ptype,
              p.free = apply(Z,2,mean),
              p.HW = EMMax.p.HW(Z))
  p
}
#Finding Beta's
#
#
EMMaxBeta <- function(x, y, Z, mutype = 0){
  r <- ncol(Z)

  #calculate beta
  if(mutype == 0){
  mod <- lapply(1:r, function(i) lm(y ~ x, weights = as.vector(Z[,i])))
  beta <- sapply(1:r, function(i) mod[[i]]$coefficients[2])
  intercept <- sapply(1:r, function(i) mod[[i]]$coefficients[1])
  }
  else{
    intercept <- EMMax.mu.ratiodist(y,Z, mutype)
    beta <- sapply(1:r, function(i) lm(I(y-intercept[i]) ~ 0 + x, weights = as.vect
or(Z[,i]))$coefficients[1])
  }
  y.hat <- lapply(1:r, function(i) intercept[i] + x*beta[i])

  return(list(beta = beta, intercept = intercept, y.hat = y.hat))
}
# EMMax.mu.ratiodist returns the predicted mu values (0:(ng-1))
# for one of the ratiodisttype models 1:10
EMMax.mu.ratiodist <- function(y,Z, mutype=1) {
  ng  <- ncol(Z)
  nr  <- nrow(Z)
```

```
  yw  <- rep(y,ng)
  wgt <- as.vector(Z)
  x   <- rep(0:(ng-1),each=nr)

  #start values:
  startf <- 1    # the intrinsic ratio of the two signal responses
  startc <- 0.1  # the signal background parameter
  startd <- -0.1 # the quadratic (nonlinearity of signal response) parameter;
  #startd was 0; with mutype 3 a 0 causes problems and testing shows that -0.1 is n
ot worse than 0
  #lower values:
  lowf <- 0.04 #lower-upper f in 0.04-25 instead of 0.001-1000 gives better converg
ence but (very slightly) lower LL
  lowc <- 0.001 #with lowc=0, many more convergence errors
  lowd <- -0.25 #was 0; with negative lowd often infinities or convergence errors
  #with lowd=0.001 a few less convergence errors than with lowd=0, but often
  #somewhat (and in some cases much) lower likelihoods
  #upper values:
  uppf <- 25   #see lowf
  uppc <- pi/2 #gives better convergence than uppc=Inf but slightly lower LL
  #pi/4 and pi/6 go further: still better convergence but lower LL
  uppd <- 1 #was Inf, values above 1 hardly change mu values
  switch (mutype,
{ formul <- yw ~ asin(sqrt((c1+x)/(c1+x + c2+f*((ng-1)-x))))
# 1 : c1, c2, f
  start=list(c1=startc,c2=startc,f=startf); lower=c(lowc,lowc,lowf); upper=c(uppc,u
ppc,uppf) },
{ formul <- yw ~ asin(sqrt((c+x)/(c+x + c+f*((ng-1)-x))))
# 2 : c, f
  start=list(c=startc,f=startf); lower=c(lowc,lowf); upper=c(uppc,uppf) },
{ formul <- yw ~ asin(sqrt((c1+x+d1*x^2)/(c1+x+d1*x^2 + c2+f*((ng-1)-x)+d2*((ng-1)-
x)^2 )))  # 3 : c1, c2, d1, d2, f
```

```
  start=list(c1=startc,c2=startc,f=startf,d1=startd,d2=startd); lower=c(lowc,lowc,l
owf,lowd,lowd); upper=c(uppc,uppc,uppf,uppd,uppd) },
```

```
{ formul <- yw ~ asin(sqrt((c+x+d1*x^2)/(c+x+d1*x^2 + c+f*((ng-1)-x)+d2*((ng-1)-x)^
2 )))      # 4 : c, d1, d2, f

  start=list(c=startc,f=startf,d1=startd,d2=startd); lower=c(lowc,lowf,lowd,lowd);
upper=c(uppc,uppf,uppd,uppd) },

{ formul <- yw ~ asin(sqrt((c1+x+d*x^2)/(c1+x+d*x^2 + c2+f*((ng-1)-x)+d*((ng-1)-x)^
2 )))      # 5 : c1, c2, d, f

  start=list(c1=startc,c2=startc,f=startf,d=startd); lower=c(lowc,lowc,lowf,lowd);
upper=c(uppc,uppc,uppf,uppd) },

{ formul <- yw ~ asin(sqrt((c+x+d*x^2)/(c+x+d*x^2 + c+f*((ng-1)-x)+d*((ng-1)-x)^2 )
))         # 6 : c, d, f

  start=list(c=startc,f=startf,d=startd); lower=c(lowc,lowf,lowd); upper=c(uppc,upp
f,uppd) }

  )


#first we try nls with the default Gauss-Newton algorithm.

#This ususally gives a better fit than the "port" algorithm, but it fails more ofte
n

suppressWarnings( {

  success <- tryCatch( {

    res.nls <- nls(formul, start=start, weights=wgt)

    T }, error=function(x) {F} )

  if (!success) {

    #if unsuccessful we try the "port" algorithm which allows to specify lower and
upper boundaries

    res.nls <- nls(formul, start=start, weights=wgt, algorithm="port", lower=lower,
upper=upper) #werkt

  }

  mu <- predict(res.nls, data.frame(x=0:(ng-1)),type="response")

})

mu

}

#Finding Variance
```

```
#

#xh: raw intensity of the first channel

#yh: raw intensity of the second channel

#Z: a matrix with posterior probabilities for each observation

#mu: cluster centers

EMMaxVar <- function(y, Z, y.hat, vartype = 0){

  wt <- apply(Z, 2, function(x) x/sum(x))

  r <- ncol(Z)


  if(vartype == 0){

    vary <- sapply(1:r, function(i) sum(as.vector(wt[,i]) * ((y - y.hat[[i]])^2)))

  }

  if(vartype == 1){

         vary <- eqvar(y, y.hat, Z)

  }

  return(vary)

}

#Helper Function Equal Variance

eqvar <- function(y,y.hat,Z){

  Z <- as.matrix(Z)

  r <- ncol(Z)


  y.hat2 <- as.vector(unlist(y.hat))

  y2 <- rep(y, times = r)


  vary <- rep(weighted.mean((y2 - y.hat2)^2, w = as.vector(Z)), times = r)


  return(vary)

}

#Maximisation function

#
```

```
#xh: raw intensity of the first channel

#yh: raw intensity of the second channel

#Z: a matrix with posterior probabilities for each observation

EMMax <- function(x, y, Z, mutype = 0, vartype = 0, ptype = "p.free"){

  p <- EMMaxP(Z, ptype)

  beta <- EMMaxBeta(x,y,Z,mutype)

  vary <- EMMaxVar(y, Z, beta$y.hat, vartype)

  psi <- list(p = p, beta = beta$beta, intercept = beta$intercept, y.hat = beta$y.h
at, vary = vary)

  o <- order(psi$intercept)

  psi <- orderpsi(psi,o)


  psi

}

#Helper function to sort the psi

orderpsi <- function(psi, o) {

  psi$intercept <- psi$intercept[o]

  psi$vary <- psi$vary[o]

  psi$p <- psi$p[o]

  psi$beta <- psi$beta[o]

  psi$y.hat <- psi$y.hat[o]


  psi

}


#EM Algorithm iterative

#

#xh: raw intensity of the first channel

#yh: raw intensity of the second channel

#ng: total number of possible clusters (one more than ploidy level)

#p.start: starting values for p
```

```
#mu.start: starting values for mu

#sd.start: starting values for the standard deviations

EMMix <- function(x, y, ng = 5, p.start, yhat.start, vary.start, maxiter = 0,mutype
= 0, vartype = 0, ptype = "p.free"){


  LLnw <- loglikf(y, p.start, yhat.start, vary.start)

  iter <- 0




  repeat { # start of EM loop


    iter <- iter+1

    LL <- LLnw


    Z <- EMGaussExp(LL$D, LL$sumD)

    cluster <- EMCluster(x, y, Z)

    psi <- EMMax(x, y, Z, mutype, vartype, ptype)

    LLnw <- loglikf(y,psi$p, psi$y.hat, psi$vary)


    if (is.na(LLnw$LL) || (max(abs(LLnw$LL - LL$LL)) < 0.0000001)||

          (maxiter>0 && iter>maxiter) ) break

  } # end of EM loop

  LL <- LLnw

  Z <- EMGaussExp(LL$D, LL$sumD)

  cluster <- EMCluster(x, y, Z)


  list(p = psi$p,intercept = psi$intercept, beta = psi$beta, vary = psi$vary, iter=
iter, LL = LL$LL, LLstart = loglikf(y, p.start, yhat.start, vary.start)$LL, Z = Z,
cluster = cluster)

}
```

```
#############################################################################
#
#Complete Bivariate Method
#
#


ConditionalMethod <- function(xh, yh, ng = 5, plotreg = T, maxiter = 0, clustertype
= "Kmeans", mutype = 1, vartype = 1, ptype = "p.free"){


  y <- asin(sqrt(yh/(xh+yh)))
  x <- (xh+yh)
  data <-  cbind(y,x)
  clustering <- switch(clustertype,
                       Mclust = ClusterInitMclust(x, y, ng),
                       Kmeans = ClusterInitKmeans(x, y, ng))


  start <- BetaInit(x, y, clustering$cluster)
  results <- EMMix(x, y, ng, p.start = clustering$p, yhat.start = start$y.hat, vary
.start = start$vary , maxiter = maxiter, mutype = mutype, vartype = vartype, ptype
= ptype)


  if(plotreg == T){
    #SimplePlot(y)
    SimplePlot2(x, y, cluster = results$cluster, intercept = results$intercept, bet
a =results$beta)
  }


  return(results)
}
SimplePlot <- function(y, nbreaks = 40, xlim=c(0,1.4)){
  nbars <- ceiling(nbreaks/(max(y,na.rm=T)-min(y,na.rm=T)))
  h <- hist(y, breaks=seq(0,1.4,by=1/nbars), plot=FALSE )
```

```
  maxy <- max(h$counts)
  h$ylim <- c(0,1.1*maxy) #so ylim will be part of the return value
  barplot (h$counts,
           width=(h$breaks[length(h$breaks)]-h$breaks[1])/(length(h$breaks)-1),
           space=0,xlim=xlim,ylim=h$ylim,col="white",
           ylab="frequency")

}
SimplePlot2 <- function(x,y, cluster, intercept, beta){
  plot(c(0,max(x)), c(0,max(y)), type = 'n', xlab = 'x', ylab = 'y', main = 'Condit
ional Method')

  for(i in 1:length(cluster)){
    points(cluster[[i]], pch=i, col = i)
    abline(a = intercept[i], b = beta[i])
  }
}
```

Appendix C: Simulation Study

```
#The number of samples from the mixture distribution

N = 1000


#Sample N random uniforms U

set.seed(2509)

U =runif(N)


#Variable to store the samples from the mixture distribution

rand.samples = matrix(NA, nrow = N, ncol = 2)


#Calculate HWE proportions

p.HWE <- function() {

  phw <- 0.5


  #HWE proportions

  c(phw^4,4*phw^3*(1-phw),6*phw^2*(1-phw)^2,4*phw*(1-phw)^3,(1-phw)^4)

}


#Store HWE proportions

HWE <- p.HWE()


#create cumulative sum

cHWE <- cumsum(HWE)


#Generate Covariance Matrix
```

```
CovarianceGenerator <- function(lamda1,lambda2,Theta){
  EigenValueMatrix <- matrix(c(lambda1,0,0,lambda2), nrow = 2, byrow = T)
  Side1 <- matrix(c(cos(Theta),-sin(Theta),sin(Theta),cos(Theta)), nrow = 2, byrow
= T)
  Side2 <- matrix(c(cos(Theta),sin(Theta),-sin(Theta),cos(Theta)), nrow = 2, byrow
= T)

  #Based on paper
  CovarianceMatrix <- Side1%*%EigenValueMatrix%*%Side2
  return(CovarianceMatrix)
}

lambda1 <- 10000
lambda2 <- 10000
Theta <- pi/4

CoMat <- CovarianceGenerator(lamda1,lambda2,Theta)

#Generate Mu's based on fitTetra linear restriction, used in the multivariate form
LinMu <- function(a0, a1, b0, b1, ng = 5, mutype = 1){

  #allele dosage
  x <- 0:(ng-1)

  #
  switch(mutype,
  {mu.alleleA <- a0 + a1*x #linear relation
  mu.alleleB <- b0 + b1*(4-x) },
  {mu.alleleA <- a0 + a1*x^2 #quadratic relation
  mu.alleleB <- b0 + b1*(4-x)^2 },
  {mu.alleleA <- ((a0 + b0)/2) + a1*x #linear relation with equal background
```

```
  mu.alleleB <- ((a0 + b0)/2) + b1*(4-x) }
  )
  cbind(mu.alleleA, mu.alleleB)
}

a0 <- 100
b0 <- 150
a1 <- 500
b1 <- 550

mu <- LinMu(a0,a1,b0,b1, mutype = 2)

#Sampling from the mixture
require(mvtnorm)
## Loading required package: mvtnorm
for(i in 1:N){
  if(U[i]<cHWE[1]){
    rand.samples[i,] = rmvnorm(1,mu[1,],CoMat)
  }else if(U[i]<cHWE[2]){
    rand.samples[i,] = rmvnorm(1,mu[2,],CoMat)
  }else if(U[i]<cHWE[3]){
    rand.samples[i,] = rmvnorm(1,mu[3,],CoMat)
  }else if(U[i]<cHWE[4]){
    rand.samples[i,] = rmvnorm(1,mu[4,],CoMat)
  }else{
    rand.samples[i,] = rmvnorm(1,mu[5,],CoMat)
  }
}

#Negative values will be removed(genotype assay can't produce negative values)
negativeCheck <- function(x){
```

```
  x[x<0]<-0
  x
}
rand.samples <- apply(rand.samples,c(1,2),negativeCheck)

#Create the classes
class <- c("genotype 1", "genotype 2", "genotype 3", "genotype 4", "genotype 5")

classAssign <- function(U, class, cHWE, N){
  genotypes = as.character(numeric(N))
  for(i in 1:N){
    if(U[i]<cHWE[1]){
      genotypes[i] = class[1]
    }else if(U[i]<cHWE[2]){
      genotypes[i] = class[2]
    }else if(U[i]<cHWE[3]){
      genotypes[i] = class[3]
    }else if(U[i]<cHWE[4]){
      genotypes[i] = class[4]
    }else{
      genotypes[i] = class[5]
    }
  }
  return(genotypes)
}

genotypes = classAssign(U, class, cHWE, N)

#Combine data and it's respective genotype
results <- data.frame(x = rand.samples[,1], y = rand.samples[,2], genotype = genoty
pes)
```

```
cluster <- lapply(1:5, function(i) data.frame(x = results[genotypes == class[i],1],
y = results[genotypes == class[i],2]))


#Density plot of the random samples

plot(density(rand.samples),main="Density Estimate of the Mixture Model")

#Plot observations color coded

plot(c(0,3000), c(0,3000), type = 'n', xlab = "x", ylab = "y", main = "Sim Study")

for(i in 1:5){

  points(cluster[[i]], pch=i, col = i)

}


#write an output file regular data

write.table(results, file = "c:/Users/Lennard/Documents/2014-2015/Internship SNPs/o
utput.txt")
```