Statistical Programming with R
Assignment 2

Tove Henning
Carl Munkby
Johannes Zetterberg

1 Least squares variance simulation

In linear regression, given that the Gauss-Markov theorem is satisfied, ordinary least squares is considered (OLS) to be the best linear unbiased estimator. If the assumption of homoscedasticity is violated, then OLS won't be the estimator with the smallest variance (the most efficient). Instead, to handle heteroscedasticity, one could make use of a weighted least squares estimator (WLS), which can be more efficient than OLS. The weights in WLS can be constructed easily if the variance is known. If the variance is unknown it is still possible to perform WLS but a certain parametric structure needs to be assumed, this method is called feasible weighted least squares (FWLS).

In the following section a comparison between the three different methods will be carried out. The goal is to see how they differ in terms of efficiency by performing estimation based on simulated data.

Ordinary least squares

To estimate the beta for the ordinary least squares estimator we used the two vector dataset and set column one to $\mathbf{y}$ and column two to $\mathbf{X}$, where $\mathbf{y}$ is the dependent variable and $\mathbf{X}$ the explanatory variable. Adding an extra intercept column of n rows to $\mathbf{X}$. Lastly a calculation of the ordinary least squares estimator has been done by using the formula: $\hat{\beta}_{OLS} = (\mathbf{XX})^{-1}\mathbf{Xy}$.

Weighted least squares

To estimate the beta for the weighted least squares estimator we used the two vector dataset and set column one to $\mathbf{y}$ and column two to $\mathbf{X}$, where $\mathbf{y}$ is the dependent variable and $\mathbf{X}$ the explanatory variable. Then an extra intercept column of n rows has been added to $\mathbf{X}$. To be able to get the error covariance matrix ω we added a zero matrix of size n x n. By running a for loop where the exponent of $\mathbf{X}$ times λ is added on the diagonal in the zero matrix calculates our ω. The value of the weighted least square estimator is then computed by: $\hat{\beta}_{WLS} = (\mathbf{X}\Omega(\lambda)^{-1}\mathbf{X})^{-1}\mathbf{X}\Omega(\lambda)^{-1}\mathbf{y}$. The function checks that λ is numeric, if not it prints "lambda is not numeric".

Feasible weighted least squares

To estimate beta for the feasible weighted least squares estimator we needed to assume some structure for the error variance depending on whether the variance form is true ($\sigma^2_{\epsilon_i} = e^{x_i\lambda}$) or false ($\sigma^2_{\epsilon_i} = 1 + x_i\lambda$). The data is divided into $\mathbf{y}$ and $\mathbf{X}$, $\mathbf{y}$ being the dependent variable, $\mathbf{X}$ being the explanatory variable. A linear regression model of $\mathbf{y}$ and $\mathbf{X}$ is estimated and the residuals from this model are used to estimate another linear regression model. In this model, the dependent variable is the natural logarithm of the squared residuals and the independent variable is $\mathbf{X}$. The coefficient of this linear model is the estimated value of λ, that is, $\hat{\lambda}$. This value of $\hat{\lambda}$ is then used to get an estimate of the error covariance matrix $\Omega(\hat{\lambda})$, where the diagonal is filled with the value of the estimated error variance for each value of x, $\hat{\sigma}^2_{\epsilon_i}$, which differs depending on whether the variance form is assumed to be true of false. The value of the feasible weighted least squares estimator is then computed by: $\hat{\beta}_{FWLS} = (\mathbf{X}\Omega(\hat{\lambda})^{-1}\mathbf{X})^{-1}\mathbf{X}\Omega(\hat{\lambda})^{-1}\mathbf{y}$.

Data

To generate data the DataFun function was generated. It generates random data with an error term dependent on the covariance structure λ and the number of observations chosen. The function starts by generating the standard deviation of epsilon since the error term is dependent on x, which comes from a uniform distribution. When all the parameters are calculated the function calculates y and then all the different values of y and x are saved in a matrix and that matrix is set to be the return value.

Simulation

The function SimFun uses all the pre-existing functions to generate data and fit models depending on the number of simulations one wants to perform. The function then returns all the fitted betas from all of the models. Except the number of simulations one can also freely choose the number of observations for the data as well, the seed number and the value of λ .

```r
### create line plot with sample size on x-axis and variance estimate on y-axis
ggplot(as.data.frame(var_obs),aes(x=as.numeric(rownames(var_obs)))) +
  geom_line(aes(y = OLS, colour = "OLS")) +
  geom_line(aes(y = WLS, colour = "WLS")) +
  geom_line(aes(y = FWLST, colour = "FWLST")) +
  geom_line(aes(y = FWLSF, colour = "FWLSF")) +
  labs(x="Number of obsvervations",y="Variance") +
  theme(legend.title = element_blank()) +
  theme(panel.grid.minor = element_blank()) +
  scale_x_continuous(breaks=c(25,50,100,200,400),limits=c(25, 400))
```

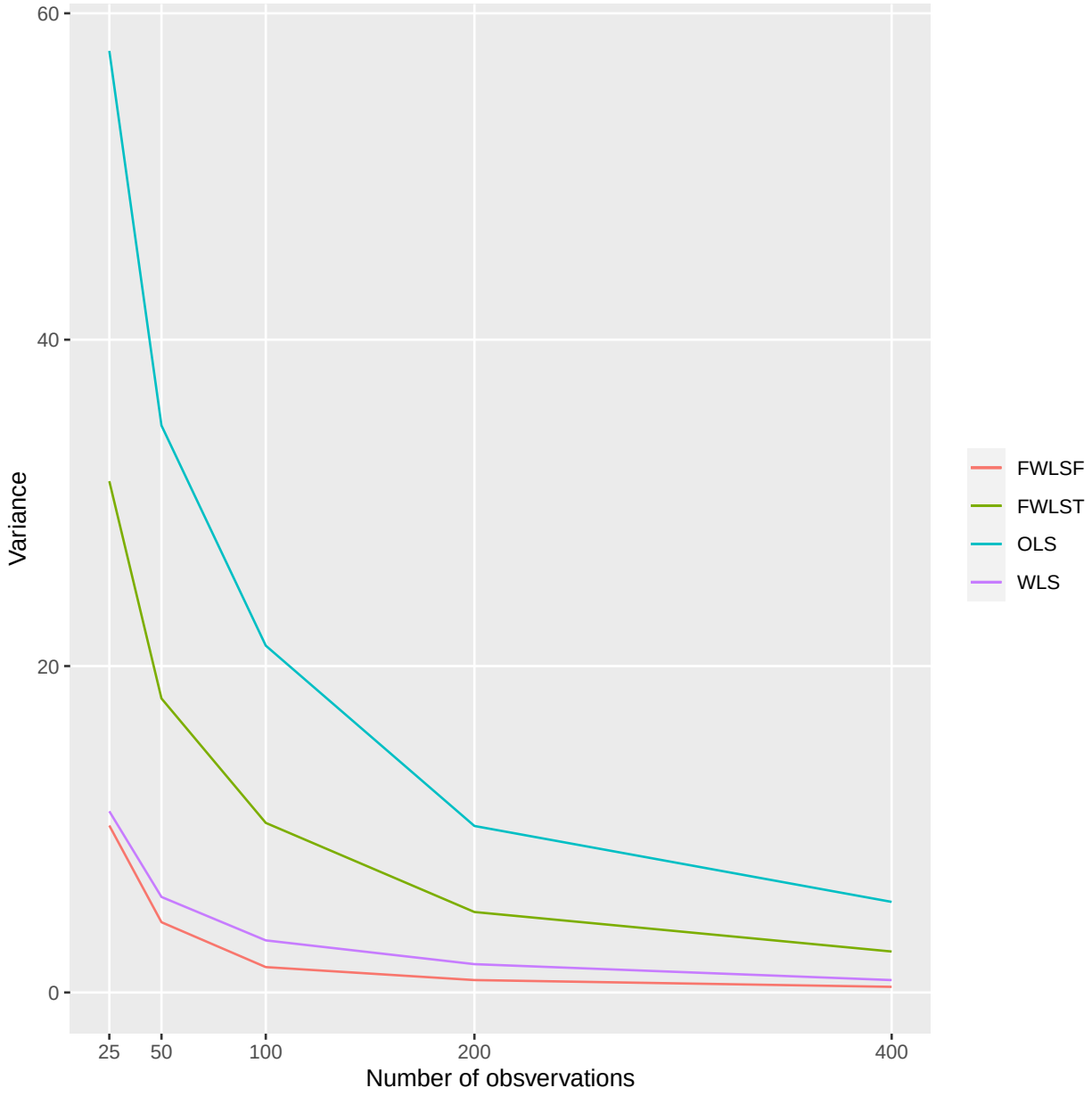


Figure 1: Variance estimates of beta for each function for different sample size

In figure 1 the variance estimates for each beta coefficient for different sample sizes is presented. We can see that the variance decreases for all beta estimates the larger the sample size is. The feasible least squares estimates the lowest variance both when the variance form is true and false, i.e. is the most efficient estimator. We can say that the FWLS is BLUE. The plot clearly shows that OLS is inefficient, it looks to lack heteroscedasticity since the variance differs much depending on sample size. WLS is better than OLS but still has some bias.

2 EM algorithm for mixture of normals

Real world data can tend to come in shapes of clusters meaning that the normal distribution won't be suitable for describing the generation of observations. However the normal distribution could still be utilized but as a linear combination with one distribution for every cluster (K). The expectation-maximation (EM) algorithm is a useful method for finding the maximum likelihood estimates for a mixture of normal models.

In the following section an example of an implementation of the EM algorithm on the dataset galaxies will be carried out. As for all clustering problems the question of the suiting number of clusters seldom has a clear cut answer and often requires further investigation.

```
# plotting the density
ggplot(galaxies, aes(x = km)) +
  geom_density()
```

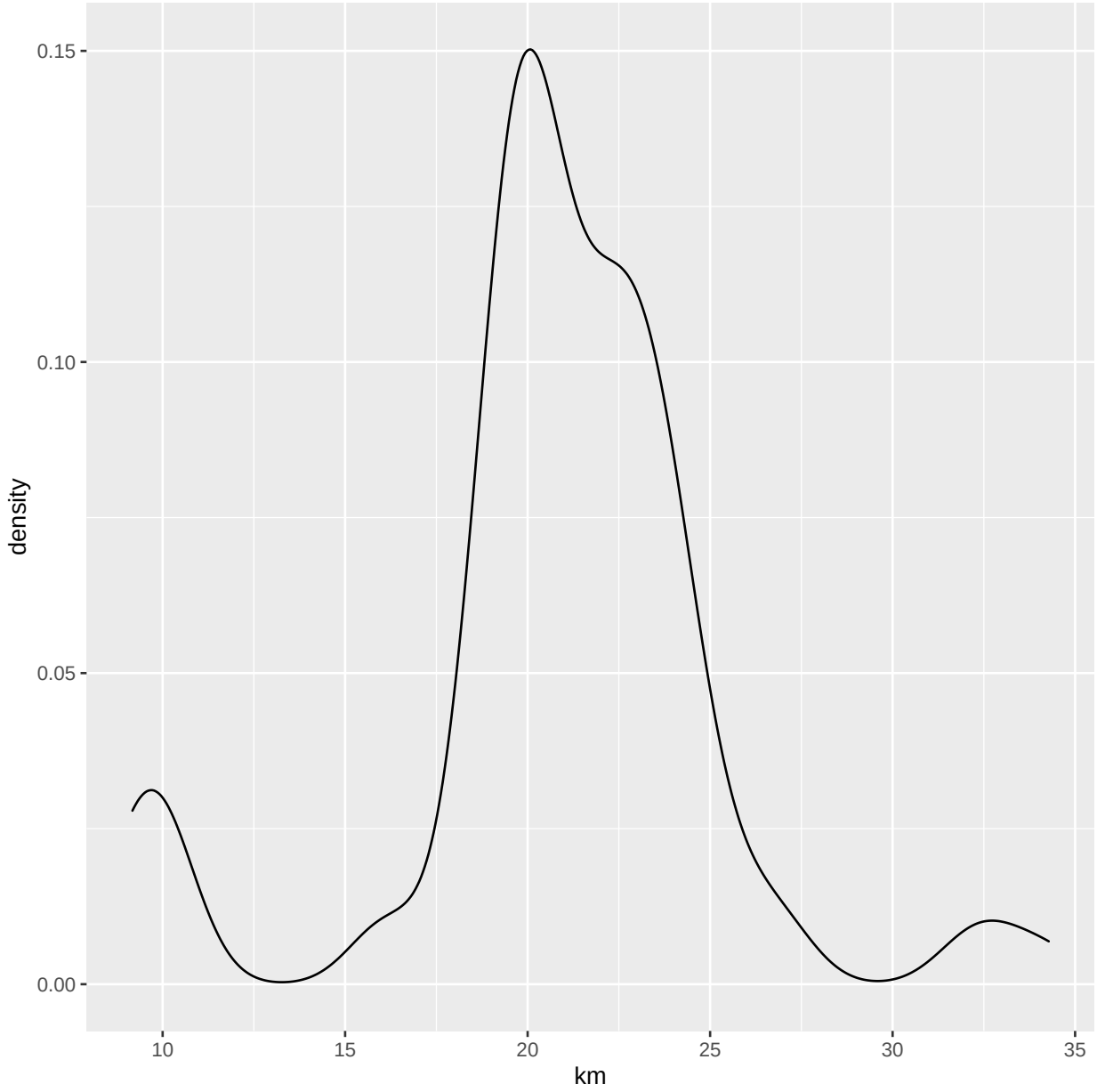


Figure 2: Distribution of the raw data

In figure 2 we can see a density plot of the data set galaxies. By examining the plot we think that three groups would be suitable to describe the data, first group that covers the area around 10 km, second in the larger area in the middle, and third that has its mean between 30 and 35 km.

Implementing the steps with functions

To be able to practically implement the EM-algorithm, several functions are created. Below follows a description of all the functions used.

gammaUpdate

The gammaUpdate function computes the E step in the EM function, that is computes the updates of $\gamma(z_{nk})$. The input of the function is a data set, in this case the galaxies data, and the values of sigma, mu and pi. The function then returns a $n \cdot K$ matrix of γz_{nk} values, where n is the number of observations in the galaxies data and K is the number of clusters. In this return, every row sums to 1.

muUpdate

The muUpdate function takes a data variable and a gamma variable as input and then calculates and returns a new μ for all of the clusters.

sigmaUpdate

The sigmaUpdate function takes a data variable and the values of gamma and mu as input and calculates and returns a new σ_k for each cluster.

piUpdate

The piUpdate function takes a gamma variable as input and then calculates and returns a new π for all clusters.

loglik

The loglik-function computes the log-likelihood of the data for given parameters. The function takes the galaxies data and value of pi, mu and sigma as input and then calculates the log-likelihood by looping through the clusters and the rows in the input data. The function then returns the log-likelihood.

initialValues

The initialValues function randomly computes the initial values for μ, σ and π and computes the initial log-likelihood. It repeats this process 100 times and only saves the values if the new log-likelihood is higher than the last one. The function returns μ, σ, π and the log-likelihood.

EM

The EM function combines all the pre-existing functions and performs the full EM-algorithm. First it initializes some starting values for μ, σ and π and computes the initial log-likelihood (step 1). It then computes γ (step 2), re-estimates the parameters (step 3) and evaluates the new log-likelihood (step 4). By the use of a while loop step 2 to step 4 is repeated until the absolute change in log-likelihood is less than 0.001. The function takes in a vector of data points and a variable with a value corresponding to the desired number of clusters. The function returns a list containing the values of log-likelihood, the mean, the standard deviation and the partitions of the clusters.

```r
### create density plot with number of K:s on x-axis and log likelihood on y-axis
ggplot(data = loglik_values) +
  geom_line(aes(x=K, y=loglik_values), color = "blue") +
  labs(x="Number of components",y= "Log likelihood")
```

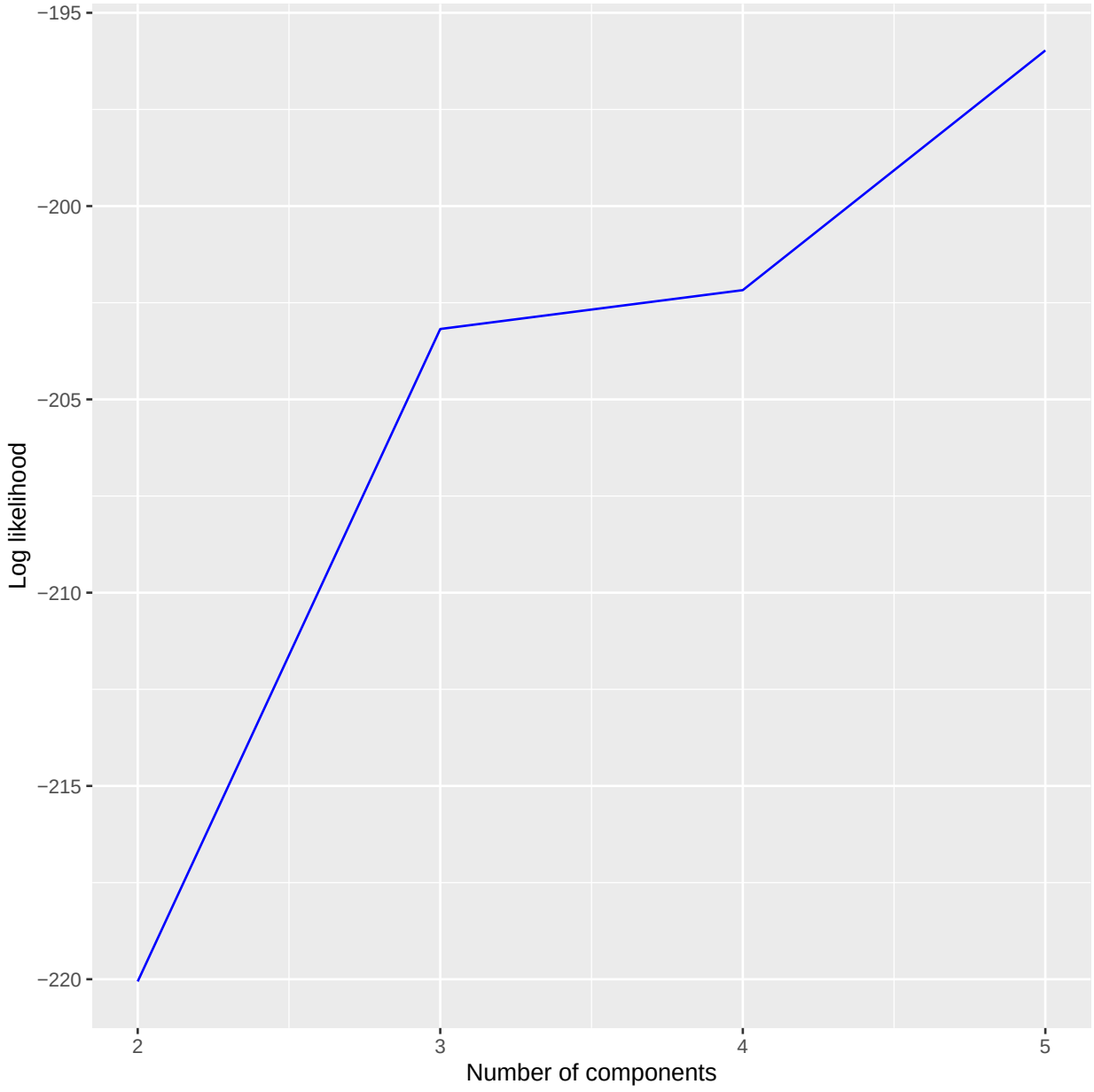
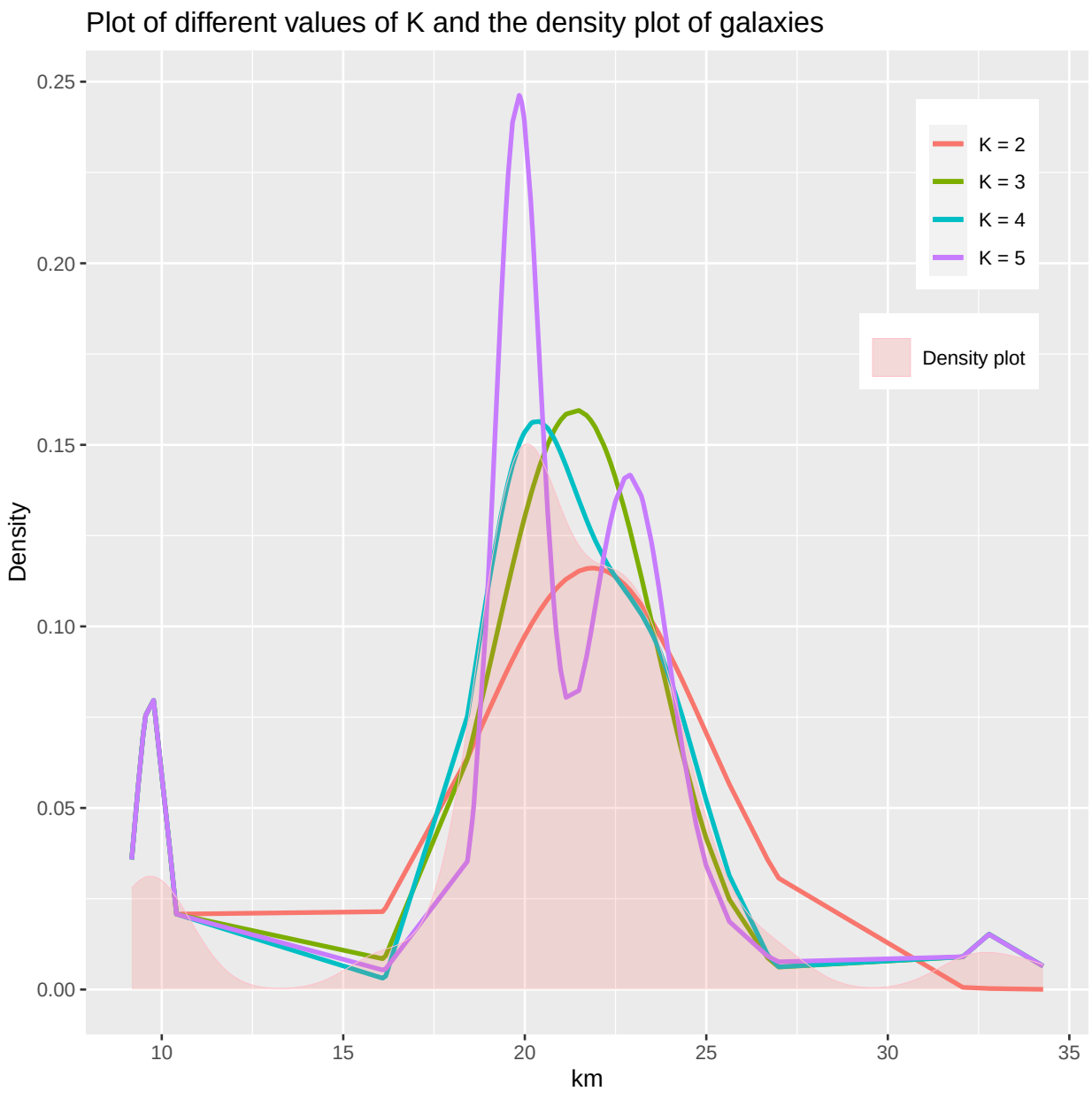


Figure 3: Log likelihood for each number of components

In figure 3 we can see the number of components and their respective log likelihoods. The results show that the log likelihood increases as the number of components K increases. It is very reasonable that the likelihood increases as the number of partitions increases but it won't be very useful to solely focus on the value of the log likelihood. Too many clusters won't be practical instead it would be more appropriate to focus on the marginal increase in log-likelihood as the number of components increases. The plot shows that the marginal increase in log-likelihood stagnates after 3 components motivating the choice of 3 clusters. However since initial values in the EM-algorithm are generated randomly the local log-likelihood optimums may differ

meaning that if we were to re-apply the EM-algorithm we would possibly observe different gradients in the plot.

```
ggplot(data = final_plot, aes(x = galaxies)) +
    geom_line(aes(y = `K = 2`, color = "K = 2"), size = 1) +
    geom_line(aes(y = `K = 3`, color = "K = 3"), size = 1) +
    geom_line(aes(y = `K = 4`, color = "K = 4"), size = 1) +
    geom_line(aes(y = `K = 5`, color = "K = 5"), size = 1) +
    geom_density(aes(fill = "Density plot"), color = "pink", alpha = 0.2, size = 0) +
    labs(x = "km", y = "Density",title="Plot of different values of K and the density plot of galaxies")
    theme(legend.title = element_blank(),legend.position = c(.95, .95),
          legend.justification = c("right", "top"),
          legend.box.just = "right",
          legend.margin = margin(6, 6, 6, 6))
```



2.1 Figure 4: Plot of different values of K and the density plot of galaxies

In figure 4 a comparison for different values of K, components, when running the EM algorithm is presented. When comparing the density plot with all the distributions the model with 4 partitions seems to fit the data best. However, when looking back at figure 3, we see that the increase in log-likelihood is very small when going from 3 to 4 components. Keeping in mind that the number of observations are quite low in this dataset, a partition of 3 is argueably more suitable since a partition of 4 or higher may run the risk of resulting in overfitting.