Statistical Programming with R
Assignment 2

Alexander Engberg & Lukas Arnroth
Department of Statistics
Uppsala University

October 5, 2020

- The assignment should be done in groups of 3 students.
- Each group must write all of its code alone. Note that it is considered cheating to share code between groups.
- Write your report in LaTeX or `knitr` (recommended).
- Your pdf file should include your code such that it is clear how your results, e.g. plots and tables, are generated. It is therefore recommended that you use `kntir`.
- Your code **must** be well documented.
- For questions which are purely about writing code in R, you can briefly outline what you did and why.
- Upload both your script (.R file) and report (.pdf file) on Studentportalen. If you use knitr, you can use the `purl()` function (in the `knitr` package) to extract all of the code.

1 Least squares variance simulation

Background. Ordinary least squares (OLS) is often used to estimate the parameters of linear regression models and is, given that the condition of the Gauss-Markov theorem are satisfied, the best linear unbiased estimator (BLUE). However, in the presence of heteroscedasticity (unequal error variances) OLS is no longer efficient (the estimator with the smallest variance). In practice, heterogeneous error variances are quite common and one way to handle them is to use a weighted least squares estimator (WLS).

WLS is applied by placing weights that are based on the structural form of the error variances onto the model before estimation, and, at least under certain forms of heteroscedasticity, WLS is more efficient than OLS. If the form of the error variances is known, constructing weights is simple. If the form is unknown, some parametric structure, here denoted λ, must be assumed and estimated, and the method is then called *feasible* WLS (FWLS).

The OLS, WLS and FWLS estimators all have analytic solutions

$$\hat{\boldsymbol{\beta}}_{OLS} = (\boldsymbol{X}'\boldsymbol{X})^{-1}\boldsymbol{X}'\boldsymbol{y}, \tag{1}$$

$$\hat{\boldsymbol{\beta}}_{WLS} = (\boldsymbol{X}'\boldsymbol{\Omega}(\lambda)^{-1}\boldsymbol{X})^{-1}\boldsymbol{X}'\boldsymbol{\Omega}(\lambda)^{-1}\boldsymbol{y}, \tag{2}$$

$$\hat{\boldsymbol{\beta}}_{FWLS} = (\boldsymbol{X}'\boldsymbol{\Omega}(\hat{\lambda})^{-1}\boldsymbol{X})^{-1}\boldsymbol{X}'\boldsymbol{\Omega}(\hat{\lambda})^{-1}\boldsymbol{y}, \tag{3}$$

where $\boldsymbol{X}$ is the $n \times p$ matrix of independent variables, $\boldsymbol{y}$ is the $n \times 1$ vector with the dependent variable, $\boldsymbol{\Omega}(\cdot)$ is the $n \times n$ error covariance matrix with either true covariance structure λ, or estimated covariance structure $\hat{\lambda}$.

For this exercise we use heteroscedastic but uncorrelated errors which means that the error covariance matrix is a diagonal matrix with the error variances on the diagonal. The variance structure used in this exercise is

$$\sigma^2_{\varepsilon_i} = e^{x_i\lambda} = e^{x_i 2},$$

so for a simple linear regression with two observations we have

$$\boldsymbol{\Omega}(\lambda) = \begin{pmatrix} e^{x_1\lambda} & 0 \\ 0 & e^{x_2\lambda} \end{pmatrix}.$$

One way to perform FWLS is to

1. Assume some structure for the error variance, here $\sigma^2_{\varepsilon_i} = e^{x_i\lambda}$.
2. Run an OLS regression on the original model $\boldsymbol{y} = \boldsymbol{X}\boldsymbol{\beta} + \boldsymbol{\epsilon}$ ($y_i = \beta_0 + \beta_1 x_i + \varepsilon_i$) and save the residuals, $\hat{u}_i$.

3. Use the squared residuals $\hat{u}_i^2$ from the regression in step 2 as an estimate of the error variance $\sigma^2_{\varepsilon_i}$. Then estimate λ by applying OLS to the equation[1]

$$\begin{aligned}\ln(\hat{u}_i^2) &= \ln(e^{x_i\lambda}e^{v}) \\ &= \lambda x_i + v,\end{aligned}$$

   where v is a well behaved error term, such that $\hat{\lambda} = (\boldsymbol{X}'\boldsymbol{X})^{-1}\boldsymbol{X}'\ln(\hat{\boldsymbol{u}}'\hat{\boldsymbol{u}})$.

4. Use $\hat{\sigma}^2_{\varepsilon_i} = e^{x_i\hat{\lambda}}$ to get an estimate of the error covariance matrix $\boldsymbol{\Omega}(\hat{\lambda}) = e^{x_i\hat{\lambda}} * \boldsymbol{I}$ that can then be inserted in the $\hat{\boldsymbol{\beta}}_{FWLS}$ estimator.

Problems. The task is to perform a small simulation study where you simulate the sampling distributions of the three estimators, estimate the variances of those distributions, and compare the variances of the estimators.

1. Write three functions, one for each estimator, with the following structures that implements the estimators using the formulae in Equations 1-3:

```r
olsFun <- function(data){
  # Your code
  return(beta_ols)
}
```

```r
wlsFun <- function(data, lambda){
  # Your code
  return(beta_wls)
}
```

```r
fwlsFun <- function(data, trueVar){
  # Your code
  return(beta_fwls)
}
```

   The input `data` should be a matrix or a data frame with two columns: one dependent and one independent variable. Add intercepts to the models inside the functions (that is, the input `data` should not include a vector of ones).

[1] The form of this equation depends on the chosen variance structure, here $e^{x_i\lambda}$.

The WLS function should have a control that checks if λ is numeric and print an informative message if not.

The FWLS-function should be implemented with both the true variance form ($\sigma^2_{\varepsilon_i} = e^{x_i \lambda}$) and with the erroneous form

$$\sigma^2_{\varepsilon_i} = 1 + x_i \lambda$$

and the logical argument `trueVar` should be `TRUE` if the correct form is used and `FALSE` otherwise.

You can test your functions with the following example data:

```r
testData <- cbind( c(0.62, 0.18, 3.92, 0.80, -5.15),
                   c(0.44, 1.49, 0.69, 0.13, 1.90) )

c(olsFun(data = testData),
wlsFun(data = testData, lambda = 2),
fwlsFun(data = testData, trueVar = TRUE),
fwlsFun(data = testData, trueVar = FALSE))


           x            x            x            x
-3.092464191  0.007392226 -2.615266161 -2.750474441
```

2. Write a function for data generation with the structure

```r
DataFun <- function(n, lambda) {
  # Your code
  # return(an n x 2 matrix or data frame)
}
```

which generates data suitable for a simple linear regression. More specifically it should generate the following:

$$\begin{aligned} x_i &\sim U(0,2), \\ \varepsilon_i &\sim N(0, e^{x_i \lambda}), \\ \beta &= 2, \\ y_i &= \beta x_i + \varepsilon_i. \end{aligned}$$

The function should take as inputs the number of observations to be generated n and the value of λ, and should output an $n \times 2$ matrix with columns representing the dependent and independent variables respectively. Although you should use $\lambda = 2$ in this exercise one should be able to change the value in the function arguments.

3. Write a simulation function with structure

   ```
   SimFun <- function(n, sim_reps, seed, lambda) {
     # Your code
     # return(Vector of four variance estimates)
   }
   ```

   which performs a simulation and should take as inputs the number of observations `n`, the number of simulation replications `sim_reps`, the `seed` (for reproducibility), and the value of λ to be passed to `DataFun` and `wlsFun`. The return should be a vector with four variance estimates, one for each estimator including FWLS both with correct and erroneous variance structure.

4. Run the simulation for sample sizes $n \in \{25, 50, 100, 200, 400\}$ and a suitable number of simulation replications. Present the results in one figure such that is easy to compare the variances estimates.

   What conclusions can you draw from the plot regarding the efficiency of the estimators? Also, exactly what is it that you have estimated in the simulation?

2 EM algorithm for mixture of normals

Background. Although the normal distribution has important analytical properties and is one of our most important tools as statisticans, it might not suffice to model real data. Clustering is a common feature of data which a simple normal distribution would fail to capture, whereas a linear combination of normal distributions would do better. A linear combination of K normal distributions are referred to as a mixture of normals,

$$p(x) = \sum_{k=1}^{K} \pi_k \mathcal{N}(x; \mu_k, \sigma_k^2), \tag{4}$$

where $\mathcal{N}(x; \mu, \sigma^2)$ is the pdf of the normal distribution with mean μ and variance σ^2 evaluated at x. Each normal density $\mathcal{N}(x; \mu_k, \sigma_k)$ in (4) is called a component, and has its own mean and variance. The parameters π_k in (4) are called mixing coefficients where $\pi_k \in [0, 1]$ and $\sum_{k=1}^{K} \pi_k = 1$ to ensure that (4) is a proper density function.

Next, we introduce a K-dimensional binary random variable $\mathbf{z} = (z_1, \ldots, z_K)$ which has 1 in one of its indices and 0 elsewhere. The values z_k therefore satisfies $z_k \in \{0, 1\}$ and $\sum_{k=1}^{K} z_k = 1$. We will consider $p(\mathbf{x}, \mathbf{z}) = p(\mathbf{z})p(\mathbf{x}|\mathbf{z})$, which is sometimes referred to as a augmented likelihood as we have included a latent variable in our model. The marginal distribution $p(\mathbf{z})$ is specified in terms of the mixing coefficients in (4), such that

$$p(z_k = 1) = \pi_k.$$

Based on how we defined $\mathbf{z}$, we can express the joint distribution of all z_k as

$$p(\mathbf{z}) = \prod_{i=1}^{k} \pi_k^{z_k}.$$

Similarly, the conditional distribution of x for a given value $\mathbf{z}$ can be expressed as

$$p(x|\mathbf{z}) = \prod_{i=1}^{K} \mathcal{N}(x; \mu_k, \sigma_k)^{z_k}. \tag{5}$$

To see that our new setup is valid, we can obtain (4) from (5) by marginalizing out $\mathbf{z}$ from $p(x, \mathbf{z})$,

$$p(x) = \sum_{\mathbf{z}} p(\mathbf{z})p(x|\mathbf{z}) = \sum_{k=1}^{K} \pi_k \mathcal{N}(x|\mu_k, \sigma_k).$$

We have thus been able to find a equivalent formulation of our first definition using the latent variable $\mathbf{z}$. It might seem like we have not gained anything but we can now work with $p(\mathbf{x}, \mathbf{z})$ instead of $p(\mathbf{x})$ which makes estimation procedures easier.

Estimation. The expectation-maximization (EM) algorithm is a powerful method for finding maximum likelihood solutions for models with latent variables. The log-likelihood of a random sample of size N from a mixture of normals is given by

$$\ell = \log p(\boldsymbol{x}|\boldsymbol{\mu}, \boldsymbol{\sigma}) = \sum_{n=1}^{N} \ln \left[\sum_{k=1}^{K} \pi_k \mathcal{N}(x_n|\mu_k, \sigma_k) \right]. \tag{6}$$

To get maximum likelihood estimates, we take the derivative with respect to μ_k and σ_k,

$$\frac{\partial \ell}{\partial \mu_k} = \sum_{n=1}^{N} \overbrace{\frac{\pi_k \mathcal{N}(x_n|\mu_k, \sigma_k)}{\sum_j \pi_j \mathcal{N}(x_n|\mu_j, \sigma_j)}}^{\gamma(z_{nk})} \frac{x_n - \mu_k}{\sigma_k^2}$$

$$\frac{\partial \ell}{\partial \sigma_k^2} = \sum_{n=1}^{N} \gamma(z_{nk}) \frac{1}{2\sigma_k^2} \left(\frac{(x_n - \mu_k)^2}{\sigma_k^2} - 1 \right)$$

The first order condition then gives

$$\frac{\partial \ell}{\partial \mu_k} = 0 \Rightarrow \mu_k = \frac{1}{N_k} \sum_{n=1}^{N} \gamma(z_{nk}) x_n$$

$$\frac{\partial \ell}{\partial \sigma_k} = 0 \Rightarrow \sigma_k = \frac{1}{N_k} \sum_{n=1}^{N} \gamma(z_{nk}) (x_n - \mu_k)^2,$$

where $N_k = \sum_{n=1}^{N} \gamma(z_{nk})$. To maximize the log likelihood with respect to the mixing coefficients, π_k, we need to use a Lagrangian multiplier to ensure $\sum_k \pi_k = 1$. The objective function is extended as

$$g(\pi_k|x) = \log p(\boldsymbol{x}|\boldsymbol{\mu}, \boldsymbol{\sigma}) + \lambda \left(\sum_{k=1}^{K} \pi_k - 1 \right),$$

which gives

$$\sum_{n=1}^{N} \gamma(z_{nk}) + \lambda = 0 \Rightarrow \pi_k = \frac{N_k}{N},$$

which is obtained by multiplying both sides by π_k, summing over k and using the constraint which gives $\lambda = -N$.

To estimate the parameters of the mixture model, we first choose initial values for the component parameters and mixing coefficients. We then alternate between a expectation step, or E step, and a maximization step, or M step. You can loosely think of it as

$$\overbrace{p(\boldsymbol{z})}^{\text{E step}}\overbrace{p(\boldsymbol{x}|\boldsymbol{z})}^{\text{M step}}.$$

Hence the reason for introducing the latent variables $\boldsymbol{z}$ in the first place! In the E step we evaluate the posterior probabilities $\gamma(z_k)$. In the M step we use the probabilities computed in the E step to re-estimate the means, variances and mixing coefficients. This alternating procedure is deterministic in the sense that a E and M step is guaranteed to increase the log likelihood function. The steps of the EM-algorithm for a mixture of K normals is outlined below

EM algorithm.

1. Initialize the means μ_k, variances σ_k and mixture parameters π_k. Compute the initial log likelihood.

2. **E step**. Compute

$$\gamma(z_{nk}) = \frac{\pi_k \mathcal{N}(x_n|\mu_k, \sigma_k)}{\sum_j \pi_j \mathcal{N}(x_n|\mu_j, \sigma_j)}, \; k = 1, \ldots, K, \; n = 1, \ldots, N \qquad (7)$$

   at current values μ_k, σ_k and π_k.

3. **M step**. Re-estimate parameters using the current probabilities and

parameter values from the previous step,

$$\mu_k^{\text{new}} = \frac{1}{N_k} \sum_{i=1}^{N} \gamma(z_{nk}) x_n, \;\; k = 1, \dots, K \tag{8}$$

$$\sigma_k^{\text{new}} = \frac{1}{N_k} \sum_{n=1}^{N} \gamma(z_{nk}) (x_n - \mu_k^{\text{new}})^2, \;\; k = 1, \dots, K \tag{9}$$

$$\pi_k^{\text{new}} = \frac{N_k}{N}, \;\; k = 1, \dots, K \tag{10}$$

4. Evaluate the log likelihood $\ln p(\boldsymbol{x}|\boldsymbol{\mu}^{\text{new}}, \boldsymbol{\sigma}^{\text{new}}, \boldsymbol{\pi}^{\text{new}})$

Steps 2-4 are repeated until the either the change in log likelihood or the parameters are sufficiently small.

Data. The data you will be working with in this task is velocities in km/sec of 82 galaxies, that has been divided by 1000. Multimodality is a common topic of interest in astronomical data, where the components correspond to some phenomena of interest. The data is part of the `StatProg` package you where asked to download during the lectures, and was recently added as `galaxies`. Unless you installed the package recently, your installed version will not have the dataset. To get the latest version of the package run

```
devtools::install_github("lukketotte/StatProg")
```

This should generally force a update of the package, but might not always work in Windows. Most problems can be fixed by manually removing the package before running the above code. If that doesn't work, you can also get the galaxies data directly from github by

```
rda_url = paste("https://github.com/lukketotte/",
                "StatProg/tree/master/data/galaxies.rda", sep ="")

download.file(rda_url, 'galaxies.rda')
```

This will download and save the galaxies data as *galaxies.rda* in the directory returned by `getwd()`. Similarly, you can visit the github page and download it manually by clicking the download button in the link.

If you updated the `StatProg` package, you can check the documentation by `?galaxies` to find out more about the data.

Problems.

1. Plot the data as a density plot. Give a suggestion on how many components are suitable based on the plot, i.e. how many groups would you say are needed to sufficiently describe the data?

2. Next you will create the functions that updates $\gamma(z_k)$, μ_k, σ_k and π_k for any number of components. This can be done either by including number of components as an argument, or keep track of it dynamically by e.g. checking the length of the mean vector.

   (a) Implement a function that computes the updates of $\gamma(z_{nk})$ in (7). The return object should be a $(n \times K)$ matrix, where K is the number of components in the mixture and with sums over rows being equal to 1.

   ```
   gammaUpdate = function(x, mu, sigma, pi){
     # your code to implement E step
   }
   ```

   where `x` is a vector of length n and `mu`, `sigma` and `pi` are vectors of length K. To simplify the calculations of the denominator in (7), keep in mind that you can use vectors for the mean and sd parameters of `dnorm()`, for example

   ```
   x = 1
   mu = c(1, 2, 3)
   sigma = c(1, 2, 3)
   dnorm(x, mu, sigma)

   [1] 0.3989423 0.1760327 0.1064827
   ```

   returns the vector $\Big(\mathcal{N}(1;1,1),\ \mathcal{N}(1;2,4),\ \mathcal{N}(1;3,9)\Big)$ without any for loop.

   (b) Implement the updates in the M step, i.e. (8), (9) and (10), using the matrix `gamma` returned from `gammaUpdate()`

   ```
   # gamma: matrix, n times K
   muUpdate = function(x, gamma, sigma){
     # your code here
   }
   sigmaUpdate = function(x, gamma, mu){
     # your code here
   }
   piUpdate = function(gamma){
     # your code here
   }
   ```

You can use the below code as a sanity check to see that your code is working as it should.

```r
library(StatProg)

mu = c(10, 20, 30)
sigma = c(2, 2, 2)
probs = c(1/3, 1/3, 1/3)

resp = gammaUpdate(galaxies, mu, sigma, probs)
mu = muUpdate(galaxies, resp, sigma)
sigma = sigmaUpdate(galaxies, resp, mu)
probs = piUpdate(resp)

cat("mu:", mu,
    "\nsigma:", sigma,
    "\nprobs:", probs,
    "\nresp[1,]", resp[1,])

mu: 9.813276 21.12324 28.68597
sigma: 0.9105702 1.921966 3.732547
probs: 0.08675691 0.8225157 0.09072738
resp[1,] 0.9999995 4.702504e-07 3.071118e-24
```

3. Implement a function that calculates (6) for given parameter estimates

```r
loglik = function(x, pi, mu, sigma){
  # your code here
}
```

4. We now have the parts needed for the EM-algorithm. Last thing is how to get the initial parameter estimates to get the algorithm started. That code is provided to you below. You can take a second to go through the code to see how it works, and note that it uses your `loglik()` function. The main idea is to randomly generate candidates for a fixed number of repetitions and simply keep the parameters that maximize the log likelihood of all proposals.

```r
initialValues = function(x, K, reps = 100){
  mu = rnorm(K, mean(x), 5)
  sigma = sqrt(rgamma(10, 5))
  p = runif(K)
  p = p/sum(p)
  currentLogLik = loglik(x, p, mu, sigma)

  for(i in 1:reps){
    mu_temp = rnorm(K, mean(x), 10)
    sigma_temp = sqrt(rgamma(10, 5))
    p_temp = runif(K)
    p_temp = p_temp/sum(p_temp)
    tempLogLik = loglik(x, p_temp, mu_temp, sigma_temp)

    if(tempLogLik > currentLogLik){
      mu = mu_temp
      sigma = sigma_temp
      p = p_temp
      currentLogLik = tempLogLik
    }
  }

  return(list("mu" = mu, "sigma" = sigma, "p" = p))
}
```

where `K` is the number of components in your mixture. Now, implement the function

```
EM = function(x, K, tol = 0.001){
  inits = initialValues(x, K, 1000)
  mu = inits$mu
  sigma = inits$sigma
  prob = inits$p

  # your code here

  return(list('loglik' = logLik, 'mu' = mu, 'sigma' = sigma, 'prob' = prob))
}
```

In your implementation, you will keep reiterating the update steps until the difference in log likelihood is less than `tol`, so you will need to keep track of the log likelihood in the previous step. Once the difference is sufficiently small (you can use the default value 0.001), the loop can be terminated and the results returned. It might look something like this

```
while(loglikDiff < tol){
  # your code
  loglikDiff = abs(prevLoglik - currentLogLik)
}
```

5. Run your EM-algorithm for $K = 2, 3, 4, 5$ to estimate the parameters using the galaxy data and compare the log likelihoods, does this result agree with your suggestion on the first problem? Lastly, you should plot (4) for the different values of K and compare to the density plot from problem 1 (all in the same plot such that it allow comparison). Keep in mind that you can take the exponent of your log likelihood that you implemented!