Diagnostics and partial residuals in a linear model

Brice Ozenne

March 8, 2021

Summary

In the document we provide a brief introduction of R (sections 1, 2, 3) and to the linear model (sections 4). We show how some of the underlying hypotheses can be checked and what to do when there is evidence that one or several assumptions are not met. We also introduce the notion of partial residuals (sections 5) and explain how to compute and display them.

1 Software

The **R** software can be downloaded at https://cloud.r-project.org/. R studio provide a convenient user interface that can be downloaded at https://www.rstudio.com/products/rstudio/. In this document the **R** code will be display in boxes:

```
1+1 ## comment about the code
```

```
[1] 2
```

while the **R** output will be displayed in dark red below the box.

When starting a fresh **R** session, only the core functionalities of **R** are available. Additional functionalities called packages can be downloaded from the CRAN using the command `install.packages`:

```
install.packages(pkgs = c("lava","car","nlme","ggfortify", "exact2x2",
     "devtools","reshape2","Publish","officer"))
```

Two of the packages we need are not available on CRAN but only on Github, this is why we also run [1]:

```
devtools::install_github("bozenne/butils")
devtools::install_github("kkholst/gof")
```

After having installed the packages, one needs to load them using the command `library` to use them in the current **R** session:

```
library(lava) ## simulate data, latent variable models
library(car) ## miscellaneous sfunction
library(nlme) ## mixed models
library(ggfortify)  ## graphical display
library(butils)  ## miscellaneous function
library(reshape2) ## wide to long format
library(gof)  ## diagnostic tests
library(Publish)  ## table 1
library(officer)  ## export to word
library(exact2x2)  ## compare proportions
```

[1] if you do not manage to install it skip that part, you should still be able to run most of the code used in this document

2 Data

2.1 Generation

We will use the dataset generated by the following commands:

```
set.seed(10)
m.lvm <- lvm(Y1[100:sigma21]~beta*AgeC+BMI2+Gene,
      Y2[110:sigma22]~beta*AgeC+BMI2+Gene,
      Y3[120:sigma23]~beta*AgeC+0.5*BMI2+Gene)
categorical(m.lvm, K = 3) <- ~Gene
distribution(m.lvm, ~Age) <- uniform.lvm(20,50)
distribution(m.lvm, ~BMI) <- gaussian.lvm(mean = 24, sd = 3)
transform(m.lvm, AgeC~Age) <- function(x, ...){x-35}
transform(m.lvm, Id~Age) <- function(x, ...){1:NROW(x)}
transform(m.lvm, BMI2~BMI) <- function(x, ...){(x-24) + (x-24)^2}
latent(m.lvm) <- ~AgeC+BMI2

p1 <- c(beta = 1, sigma21 = 1, sigma22 = 2, sigma23 = 3)
p2 <- c(beta = -1, sigma21 = 4, sigma22 = 4, sigma23 = 4)
d <- rbind(cbind(lava::sim(n = 1e2, m.lvm, latent=FALSE, p = p1), Gender = "Male"
    ),
    cbind(lava::sim(n = 1e2, m.lvm, latent=FALSE, p = p2), Gender = "Female")
    )

d$Gender <- as.factor(d$Gender)
d$Gene <- factor(d$Gene, labels = c("A","B","C"))
d$Y1 <- round(d$Y1,1)
d$Y2 <- round(d$Y2,1)
d$Age <- round(d$Age,1)
d$BMI <- round(d$BMI,1)
d <- d[,c("Id","Age","Gender","BMI","Gene","Y1","Y2","Y3")]
head(d)
```

```
  Id  Age Gender  BMI Gene    Y1    Y2       Y3
1  1 44.2   Male 23.8    A 109.1 120.8 131.7429
2  2 41.3   Male 27.5    B 123.2 133.9 136.3850
3  3 27.4   Male 30.6    A 140.6 154.0 136.1408
4  4 35.3   Male 25.2    C 104.4 116.2 125.0175
5  5 37.1   Male 21.8    A 105.0 113.2 123.6257
6  6 29.9   Male 18.1    C 125.9 136.2 131.7966
```

We will then export the data in a .csv format using

```
write.csv(d, "data.csv", row.names = FALSE)
```

Note: in "real life" studies, this step does not exist. Instead an experiment is performed where some data are collected.

2.2 Data management

2.2.1 Working directory

The working directory is where R, by default, look for files to import and export data or figures. The current working directory can be accessed using:

```
path <- getwd()
path
```

```
[1] "/home/brice/Documents/GitHub/bozenne.github.io/doc/2020-09-17-linearModel"
```

It can be changed using the function `setwd()`:

```
path2 <- "~"
setwd(path2)
```

We can check that the working directory has indeed changed calling again `getwd()`:

```
getwd()
```

```
[1] "/home/brice"
```

We move back to the original working directory doing:

```
setwd(path)
```

2.2.2 Importing the data

It is a good idea to start by checking that the working directory contains the data we want to import. For instance the file `data.csv` is storing the data, we can use:

```
file.exists("data.csv")
```

```
[1] TRUE
```

We can also list all files in the current directory with a `.csv` extension using:

```
list.files(pattern = ".csv")
```

```
[1] "data.csv"
```

We can also display the first lines of the file using:

```
readLines("data.csv")[1:3]
```

```
[1] "\"Id\",\"Age\",\"Gender\",\"BMI\",\"Gene\",\"Y1\",\"Y2\",\"Y3\""
[2] "1,44.2,\"Male\",23.8,\"A\",109.1,120.8,131.742898261202"
[3] "2,41.3,\"Male\",27.5,\"B\",123.2,133.9,136.385038040154"
```

We can see that the columns are separated with `,` and that the `.` indicates the decimal values. Moreover the words such as the columns names or the subject identities are surrounded by `\"` (e.g. `\"Id\"` stand for Id). Finally in this example there is no missing values but if there was it is important to know how they are encoded.

The command to import the data depends on the type of file. Here for a `.csv` file we use `read.csv`. Luckily the default arguments `sep`, `dec`, `quote` are correctly specified:

```
args(read.csv)
```

```
function (file, header = TRUE, sep = ",", quote = "\"", dec = ".",
    fill = TRUE, comment.char = "", ...)
NULL
```

The argument `header` set to `TRUE` indicates that the first line of the dataset contains the column names (and not the actual data). The `...` indicates there are additional arguments that are not shown here (see the documentation using `help(read.csv)`). For instance, in presence of missing values, one would need to specify the argument `na.string`. Here it is sufficient to do:

```
dfW <- read.csv("data.csv")
```

Other functions exists to import other types of data, e.g. `read.table` for `.txt` files, `read.xlsx` from the xlsx package for `.xlsx` file, or `read.spss` from the foreign package for spss data files. One should always inspect if R has correctly imported the data, e.g. using:

```
str(dfW)
```

```
'data.frame':        200 obs. of  8 variables:
 $ Id    : int  1 2 3 4 5 6 7 8 9 10 ...
 $ Age   : num  44.2 41.3 27.4 35.3 37.1 29.9 33.1 26 43.6 43.5 ...
 $ Gender: Factor w/ 2 levels "Female","Male": 2 2 2 2 2 2 2 2 2 2 ...
 $ BMI   : num  23.8 27.5 30.6 25.2 21.8 18.1 18.1 21.2 27.6 22.1 ...
 $ Gene  : Factor w/ 3 levels "A","B","C": 1 2 1 3 1 3 3 1 3 3 ...
 $ Y1    : num  109 123 141 104 105 ...
 $ Y2    : num  121 134 154 116 113 ...
 $ Y3    : num  132 136 136 125 124 ...
```

In this example, the two columns contain character strings (`Factor` is a type of character strings in R) and the rest contains numerical values.

2.2.3 Data processing

Often the raw data needs to be transformed before being analyzed:

- A typical example is when one need to deal with the variable:

```
gender <- c(1,0,1,0,1) ## what is 1? what is 0?
```

This is already better:

```
female <- c(1,0,1,0,1) ## we can guess that 1: female and 0: male
```

but it is a good practice in such situation to rename the actual values into something understandable:

```
factor(gender, levels = 0:1, labels = c("Female","Male"))
```

```
[1] Male   Female Male   Female Male
Levels: Female Male
```

- With repeated measurements per individual, one often needs to reshape his dataset from the wide format (one line per individual) to the long format (one line per measurement). This can be done using the `melt` method. The opposite operation can be performed using `dcast`.

```
str(dfW)
```

```
'data.frame':        200 obs. of  8 variables:
 $ Id    : int  1 2 3 4 5 6 7 8 9 10 ...
 $ Age   : num  44.2 41.3 27.4 35.3 37.1 29.9 33.1 26 43.6 43.5 ...
 $ Gender: Factor w/ 2 levels "Female","Male": 2 2 2 2 2 2 2 2 2 2 ...
 $ BMI   : num  23.8 27.5 30.6 25.2 21.8 18.1 18.1 21.2 27.6 22.1 ...
 $ Gene  : Factor w/ 3 levels "A","B","C": 1 2 1 3 1 3 3 1 3 3 ...
 $ Y1    : num  109 123 141 104 105 ...
 $ Y2    : num  121 134 154 116 113 ...
 $ Y3    : num  132 136 136 125 124 ...
```

```
dfL <- reshape2::melt(dfW, id.vars = c("Id","Gender","Age","BMI","Gene"),
        measure.vars = c("Y1","Y2","Y3"),
        value.name = "score",variable.name = "outcome")
head(dfL)
```

```
  Id Gender  Age  BMI Gene outcome score
1  1   Male 44.2 23.8    A      Y1 109.1
2  2   Male 41.3 27.5    B      Y1 123.2
3  3   Male 27.4 30.6    A      Y1 140.6
4  4   Male 35.3 25.2    C      Y1 104.4
5  5   Male 37.1 21.8    A      Y1 105.0
6  6   Male 29.9 18.1    C      Y1 125.9
```

- It is often a good idea to restrict the dataset to the relevant variables (e.g. remove genetic data if they are not of interest). It is easier to work with and to display in the next steps. This can for instance be done by defining the variables of interest:

```
keep.var <- c("Id","BMI","Y1")
```

We can check that the variables defined in `keep.var` are in `df`:

```
keep.var %in% names(dfW)
```

```
[1] TRUE TRUE TRUE
```

and then subset the initial dataset:

```
dfW.red <- dfW[,keep.var]
head(dfW.red)
```

```
  Id  BMI    Y1
1  1 23.8 109.1
2  2 27.5 123.2
3  3 30.6 140.6
4  4 25.2 104.4
5  5 21.8 105.0
6  6 18.1 125.9
```

- Often after having imported the data we want to change its column names. First we need to know the current column names. The `names` function can be used to output all the column names:

```
names(dfW)
```

```
[1] "Id"     "Age"    "Gender" "BMI"    "Gene"   "Y1"     "Y2"     "Y3"
```

Alternatively the `grep` function will output any column name containing a given string of characters:

```
grep(pattern = "Y", x = names(dfW), value = TRUE)
```

```
[1] "Y1" "Y2" "Y3"
```

Then, we can rename columns one at a time using:

```
names(dfW)[names(dfW) == "Y1"] <- "baseline_score"
names(dfW)[names(dfW) == "Y2"] <- "followup_score"
names(dfW)[names(dfW) == "Y3"] <- "final_score"
head(dfW)
```

```
  Id  Age Gender  BMI Gene baseline_score followup_score final_score
1  1 44.2   Male 23.8    A          109.1          120.8    131.7429
2  2 41.3   Male 27.5    B          123.2          133.9    136.3850
3  3 27.4   Male 30.6    A          140.6          154.0    136.1408
4  4 35.3   Male 25.2    C          104.4          116.2    125.0175
5  5 37.1   Male 21.8    A          105.0          113.2    123.6257
6  6 29.9   Male 18.1    C          125.9          136.2    131.7966
```

To rename several columns at the same time we can use:

```
old2new <- c("baseline_score" = "Y1",
      "followup_score" = "Y2",
      "final_score" = "Y3")
names(dfW)[match(names(old2new),names(dfW))] <- old2new
head(dfW)
```

```
  Id  Age Gender  BMI Gene    Y1    Y2       Y3
1  1 44.2   Male 23.8    A 109.1 120.8 131.7429
2  2 41.3   Male 27.5    B 123.2 133.9 136.3850
3  3 27.4   Male 30.6    A 140.6 154.0 136.1408
4  4 35.3   Male 25.2    C 104.4 116.2 125.0175
5  5 37.1   Male 21.8    A 105.0 113.2 123.6257
6  6 29.9   Male 18.1    C 125.9 136.2 131.7966
```

Other useful functions are `tolower` to convert characters to lower case and `gsub` to remove a specific pattern in a character vector, e.g.:

```
gsub(pattern = ".", replacement = "", x = c("a..","b..."), fixed = TRUE)
```

```
[1] "a" "b"
```

Many of the other data processing steps are specific to each study and we won't discuss them in this document.

3 Descriptive statistics

Before doing any analysis, it is a good practice to describe the data that are to be analyzed. The has several aims:

- **check that that database contains the population of interest**, i.e. individuals in the database are indeed those the we want to study and we have all of them.
- **check that the collected values are plausible**, e.g. if the inclusion criteria include that the age range is between 18 and 99 years, then one should check that this is indeed the case.
- **check that the collected values are coded as expected**, e.g. age is usually coded in years (and not in months).
- **check that the collected values are distributed as expected**, e.g. is there missing values? Are the values uniformly spread? Bimodal? Concentrated at low or high values?

Note: one should checks that for all the variables of interest. This can appear time-consuming but can really save you time at latter stages.

- **produce your table 1** i.e. a descriptive table of your cohort that is almost always included in an article. You can for instance use the function `univariateTable` from the Publish package:

```
myTable1 <- univariateTable(Gender ~ Age + BMI + Y1 + Y2 + Y3,
       data = dfW)
myTable1
```

```
  Variable     Level Female (n=100) Male (n=100) Total (n=200) p-value
1      Age mean (sd)     33.9 (8.7)     35.1 (9)    34.5 (8.9)  0.3459
2      BMI mean (sd)     24.2 (2.6)   23.8 (3.4)        24 (3)  0.4307
3       Y1 mean (sd)   109.1 (13.8)   112.2 (17)  110.7 (15.5)  0.1606
4       Y2 mean (sd)   119.1 (13.3)   122.4 (17)  120.7 (15.4)  0.1335
5       Y3 mean (sd)   126.1 (10.7)   127 (11.5)  126.6 (11.1)  0.5601
```

You can also export this table in a word document with the package officer:

```
myTable1.doc <- body_add_table(x = read_docx(),
          value =  summary(myTable1))
print(myTable1.doc, target = "./Table1.docx")
```

To keep the code simple, we only present here a very basic application of these tools. More complex tables with a nicer display in word can be obtain with a bit of coding.

- **make synthetic representations of your data** using graphs or images. This can be useful to visualize your data and help your collaborators to understand what you have collected or what you are trying to show.

```
library(ggplot2)
gg <- ggplot(dfL, aes(x = BMI, y = score, color = Gender, group = Gender))
gg <- gg + geom_point()
gg <- gg + facet_wrap(~outcome, labeller = label_both)
gg <- gg + geom_smooth(method = "lm", se = FALSE)
gg
```

```
`geom_smooth()` using formula 'y ~ x'
```

You can then export the figure in a folder `figures` using:

```
pdf("./figures/descriptive.pdf", width = 12)
gg + theme(text = element_text(size=25))
dev.off()
```

- **Compare percentages when considering categorical data**: the usual way to compare the distribution of a categorical variable between two groups is to run a Fisher test using `fisher.test` in the R software. It returns a p-value and an estimate of the odd ratio with its confidence interval. For instance, consider the following dataset:

```
mytable <- rbind(c(8,5), c(4,15))
dimnames(mytable) <- list(c("control","treatment"), c("-","+"))
mytable
```

```
          -  +
control   8  5
treatment 4 15
```

The Fisher test outputs:

```
fisher.test(mytable)
```

```
        Fisher's Exact Test for Count Data

data:  mytable
p-value = 0.02996
alternative hypothesis: true odds ratio is not equal to 1
95 percent confidence interval:
  0.9953576 38.7853302
sample estimates:
odds ratio
  5.622612
```

This approach admits two drawbacks:

- the p-value may not agree with the confidence interval of the odd ratio regarding the rejection of the null hypothesis
- the odd ratio is a rather complex quantity to understand.

Instead one can use the function `binomMeld.test` (package *exact2x2*) to perform a test on the proportions:

```
binomMeld.test(x1=mytable["control","+"],n1=sum(mytable["control",]),
        x2=mytable["treatment","+"],n2=sum(mytable["treatment",]),
        parmtype="difference")
```

```
        melded binomial test for difference

data:  sample 1:(5/13), sample 2:(15/19)
proportion 1 = 0.38462, proportion 2 = 0.78947, p-value = 0.05077
alternative hypothesis: true difference is not equal to 0
95 percent confidence interval:
 -0.001077177  0.715576028
sample estimates:
difference (p2-p1)
         0.4048583
```

This time the p-value is consistent with the confidence interval.

4 Introduction to the linear model

Imagine we would like to model the age effect on the outcome, but accounting for a possible gender and gene effect. In **R** we would use the `lm` function:

```
e.lm <- lm(Y1 ~ Gender + Age + Gene + BMI, data = dfW)
e.lm
```

```
Call:
lm(formula = Y1 ~ Gender + Age + Gene + BMI, data = dfW)

Coefficients:
(Intercept)   GenderMale          Age        GeneB        GeneC          BMI
    91.5610       3.7984      -0.1358       7.8329       5.8120       0.7365
```

Denote for the $i-th$ patient its outcome value by Y_i (can be any real number), its gender value by $Gender_i$ (can be "Male" or "Female"), its gene value by $Gene_i$ (can be "A", "B", or "C"), and its BMI value by BMI_i. Mathematically, this linear model can be written:

$$\begin{aligned} Y_i =& \alpha + \beta_{Gender} * \mathbb{1}_{Gender_i="Male"} + \beta_{Age} * Age_i + \beta_{GeneB} * \mathbb{1}_{Gene_i="B"} + \beta_{GeneC} * \mathbb{1}_{Gene_i="C"} \\ & + \beta_{BMI} * BMI_i + \varepsilon_i \end{aligned}$$

where $\boldsymbol{\beta} = (\alpha, \beta_{Gender}, \beta_{Age}, \beta_{GeneB}, \beta_{GeneC}, \beta_{BMI})$ is the vector of model parameters. Their value is shown just above (e.g. $\alpha = 21.3988$). Here $\mathbb{1}_{\cdot}$ denotes the indicator function taking value 1 if "." is true and 0 otherwise. ε_i is the residual error, i.e. the difference between the observed value and the fitted value. Consider for instance the first individual:

```
d[1,]
```

```
  Id  Age Gender  BMI Gene    Y1    Y2       Y3
1  1 44.2   Male 23.8    A 109.1 120.8 131.7429
```

its observed value is 109.2 and we can computed its fitted value as:

$$\begin{aligned} \hat{Y}_1 &= \widehat{\alpha} + \widehat{\beta}_{Gender} * 0 + \widehat{\beta}_{Age} 48 + \widehat{\beta}_{GeneB} * 0 + \widehat{\beta}_{GeneC} * 0 \\ &= 91.5610 + 3.7984 * 1 - 0.1358 * 44.2 + 7.8329 * 0 + 5.8120 * 0 + 0.7365 * 23.8 \\ &= 106.8857 \end{aligned}$$

Here the hat on top of the β refer to the estimated coefficient.

This can also be obtained using the `fitted` method in **R** (the discrepancy comes from rounding the coefficient values at the 4th digit):

```
fitted(e.lm)[1]
```

```
       1
106.8839
```

Often, for conciseness, this linear model can be abbreviated as:

$$Y_i = X_i\boldsymbol{\beta} + \varepsilon_i$$

where $X_i = (1, \mathbb{1}_{Gender_i="Male"}, Age_i, \mathbb{1}_{Gene_i="B"}, \mathbb{1}_{Gene_i="C"})$ and $X_i\boldsymbol{\beta}$ is the matrix product between the row vector X_i and the column vector $\boldsymbol{\beta}$. More generally, i.e. at the population level instead of the individual level, we also write $Y = X\boldsymbol{\beta} + \varepsilon$ to describe the relationship between the random variables Y, X, ε.

4.1 Assumptions

A linear model $Y = X\beta + \varepsilon$ is a model studying the effects (β) of covariates (X) on the expected value of the outcome Y. Maximum likelihood (ML) estimation leads to unbiased estimates of β if the following assumptions are satisfied:

- **(A0)**: no unobserved confounders.
- **(A1)**: $\mathbb{E}[Y_i|X] = X_i\beta$ correct specification of the functional form of the covariates.
- **(A2)**: identically distributed and **(A3)** independent residuals.
  Under the normality assumption, it simplifies to **(A2)** homoschedasticity $\mathbb{V}ar[Y_i|X] = \sigma^2$ and **(A3)** uncorrelatedness $\forall i \neq j$, $\mathbb{C}ov[Y_i, Y_j|X] = 0$.

While not needed per se, the assumption of:

- **(A4)**: normally distributed residuals is often mentioned since (i) normality of the estimates holds exactly in finite samples (instead of asymptotically) i.e. p-values/CIs are reliable even in small samples, (ii) it ensures that MLE is the best estimation procedure, (iii) checking **(A2)** and **(A3)** is simplified.

Additional assumptions (automatically fullfield under normality) are typically necessary to ensure reliable and interpretable estimates:

- **(A4-bis)**: approximately symmetric and unimodal - otherwise modeling the expected value (aka the mean value) may not be very relevant.
- **(A5)**: absence of outliers - otherwise the estimates may be very sensitive to the value of a few observations which is often undesirable.

4.2 Interpretation of the regression coefficients

If the assumptions (A1-A3), β_{age} reflect the association between age and the outcome. This means that for fixed gene, gender, and BMI, if we observe an individual A one year older than an individual B then we would also expect that its value for the outcome Y to differ by β_{age}.
If we in addition make causal assumptions (mainly A0: no unobserved confounder) then we can interpret β_{Age} as the effect of age on the outcome. This means that if we could change the age of an individual by one unit then its variation in outcome should be β_{age}.

4.3 Hypothesis testing

We want to formally test whether there is an effect of age on the outcome. We first need to make the distinction between:

- β^0_{age} the true but unknown age effect (may be 1.5)
- $\widehat{\beta}_{age}$ the estimated age effect (here 1.5326 using maximum likelihood)

We would like to test the null hypothesis of no age effect:

$$(\mathcal{H}_0)\ \beta^0_{age} = 0$$

Since the parameters are estimated by ML and assuming that the model is correctly specified, we know that the asymptotic distribution of the parameter is Gaussian. This means that for large sample size, the fluctuation of the estimated values follows a normal distribution. For instance:

$$\hat{\beta}_{age} \underset{n\to\infty}{\sim} \mathcal{N}\left(\beta^0_{age}, \sigma^2_{\beta_{age}}\right)$$

where $\sigma^2_{\beta_{age}}$ is the variance of the MLE, i.e. the uncertainty surrounding our estimation of the association. It follows that:

$$t_{\beta_{age}} = \frac{\hat{\beta}_{age} - \beta^0_{age}}{\sigma^2_{\beta_{age}}} \underset{n\to\infty}{\sim} \mathcal{N}(0,1) \tag{1}$$

So under the null hypothesis of no association between the outcome and the exposure the statistic $t_{\beta_{age}}$ should follow a standard normal distribution. Very low or very large values are unlikely to be observed and would indicate that the null hypothesis does not hold. This is called a (univariate) Wald test.

The result of this tests can be obtained using the `summary` method [2]:

```
summary(e.lm)$coef
```

```
             Estimate Std. Error   t value     Pr(>|t|)
(Intercept) 91.5609817  9.4890315  9.649139 3.049271e-18
GenderMale   3.7984283  2.1504309  1.766357 7.890869e-02
Age         -0.1358252  0.1222051 -1.111452 2.677492e-01
GeneB        7.8328783  2.6882498  2.913746 3.990606e-03
GeneC        5.8120279  2.5395657  2.288591 2.318060e-02
BMI          0.7364696  0.3583963  2.054903 4.122862e-02
```

95% confidence intervals for the model parameters can then be obtained using the `confint` method:

```
confint(e.lm)
```

```
                 2.5 %     97.5 %
(Intercept) 72.84607298 110.275890
GenderMale  -0.44279672   8.039653
Age         -0.37684645   0.105196
GeneB        2.53093057  13.134826
GeneC        0.80332490  10.820731
BMI          0.02961616   1.443323
```

Note that based on the estimate and standard errors, we could compute the p-value ourself:

```
beta <- summary(e.lm)$coef[,"Estimate"]
sigma <- summary(e.lm)$coef[,"Std. Error"]
df <- df.residual(e.lm)
t.abs <- abs(beta/sigma)
rbind(asymptotic = 2*(1-pnorm(t.abs)),
      corrected = 2*(1-pt(t.abs, df = df)))
cat("degrees of freedom:", df,"\n")
```

```
           (Intercept) GenderMale       Age       GeneB      GeneC        BMI
asymptotic           0 0.07733600 0.2663737 0.003571198 0.02210311 0.03988840
corrected            0 0.07890869 0.2677492 0.003990606 0.02318060 0.04122862
degrees of freedom: 194
```

[2] In reality R is automatically performing a correction that improves the control of the type 1 error. Indeed we usually don't know $\sigma^2_{\beta_{age}}$ and plugging-in its estimate in equation (1) modifies the distribution of $t_{\beta_{age}}$ in small samples. The correction uses a Student's t distribution instead of a Gaussian distribution.

and the confidence intervals:

```
cbind(normLower = beta + qnorm(0.025) * sigma,
      normUpper = beta + qnorm(0.975) * sigma,
      tLower = beta + qt(0.025, df = df) * sigma,
      tUpper = beta + qt(0.975, df = df) * sigma)
```

```
              normLower   normUpper      tLower     tUpper
(Intercept) 72.96282173 110.1591416 72.84607298 110.275890
GenderMale  -0.41633879   8.0131954 -0.44279672   8.039653
Age         -0.37534289   0.1036925 -0.37684645   0.105196
GeneB        2.56400558  13.1017511  2.53093057  13.134826
GeneC        0.83457057  10.7894853  0.80332490  10.820731
BMI          0.03402571   1.4389134  0.02961616   1.443323
```

4.4 Checking assumptions made when fitting a linear model

4.4.1 (A0): no unobserved confounders

(A0) is in general impossible to check.

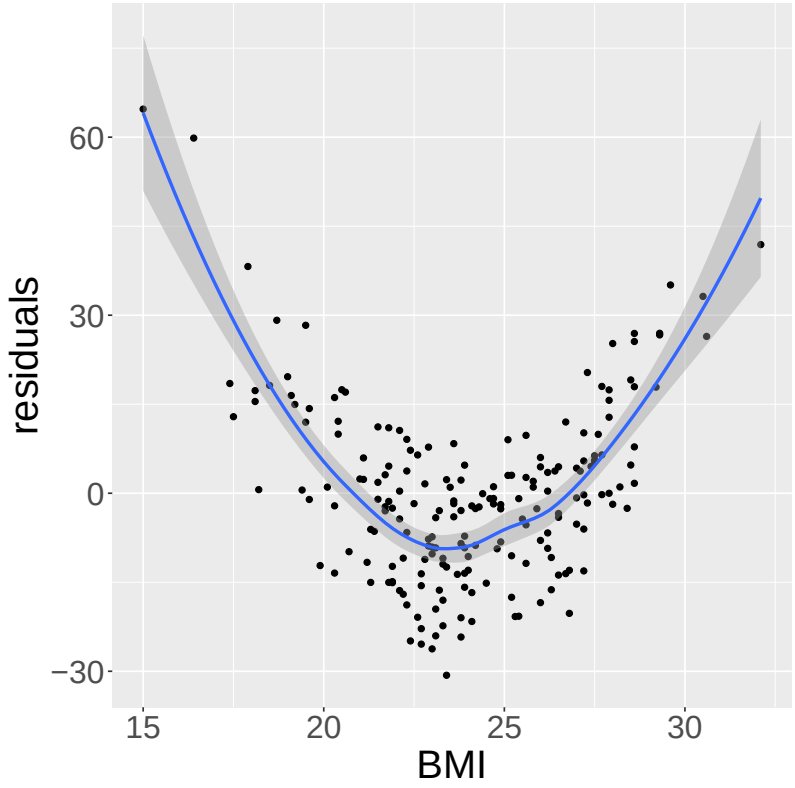
4.4.2 (A1): correct specification of the functional

(A1) can be (artificially) decomposed into two part:

- in absence of interaction, **is the effect of the continuous variables correctly modeled?** Typically it is modeled as a linear effect and the question is is there a non-linear effect. We can look at the plot of the covariate vs. the residuals and search for any trend:

```
gg <- ggplot(d, aes(x = BMI, y = residuals(e.lm)))
gg <- gg + geom_point() + geom_smooth() + ylab("residuals")
gg
```

```
`geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



(similar plots can be automatically generated using the `crPlots` or `ceresPlots` function from the car package). A p-value for testing the correct specification of the functional form for the covariate can be obtained using the `cumres` function from the gof package:

```
cumres(e.lm, variable = "BMI")
```

```
    p-value(Sup) p-value(L2)
BMI            0           0

Based on 1000 realizations.
```

Remedies: if a trend is found, a possible remedy is to use splines to model the non-linear relationship, e.g.

```
e.gam <- mgcv::gam(Y1 ~ Gender + Age + Gene + s(BMI), data = dfW)
```

In this simple example, it looks like a quadratic function of BMI would be enough:

```
e.lm.1 <- lm(Y1 ~ Gender + Age + Gene + BMI + I(BMI^2), data = dfW)
cumres(e.lm.1, variable = "BMI")
```

```
    p-value(Sup) p-value(L2)
BMI        0.268       0.444

Based on 1000 realizations.
```

Note that this type of test is not appropriate to detect missing interaction:

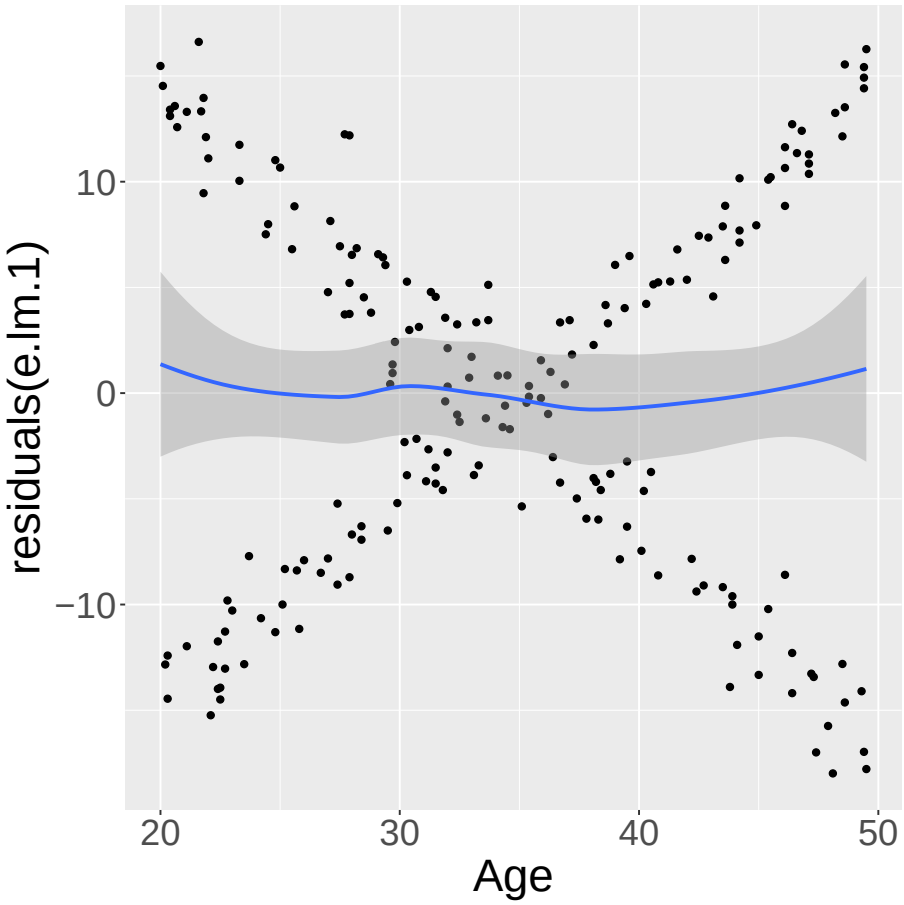
```
cumres(e.lm.1, variable = "Age")
```

```
    p-value(Sup) p-value(L2)
Age        0.074       0.768

Based on 1000 realizations.
```

while the display of the residuals can be informative

```
gg <- ggplot(dfW, aes(x = Age, y = residuals(e.lm.1))) + geom_point() +
    geom_smooth()
gg
```



- **checking for interactions** is hard because the number of possible interactions grows quickly with the number of covariates. A typical test would be to compare a model with interactions to a model without interactions:

```
e.lm.2 <- update(e.lm, Y1 ~ Gender*Age + Gene + BMI + I(BMI^2))
anova(e.lm.1, e.lm.2)
```

```
Analysis of Variance Table

Model 1: Y1 ~ Gender + Age + Gene + BMI + I(BMI^2)
Model 2: Y1 ~ Gender + Age + Gene + BMI + I(BMI^2) + Gender:Age
  Res.Df     RSS Df Sum of Sq      F    Pr(>F)
1    193 16345.2
2    192   509.8  1     15835 5963.5 < 2.2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Note that in that case a test on the cumulative residuals process would not detect any issue:

```
cumres(e.lm.1, variable = "predicted")
```

```
          p-value(Sup) p-value(L2)
predicted        0.664       0.778

Based on 1000 realizations.
```

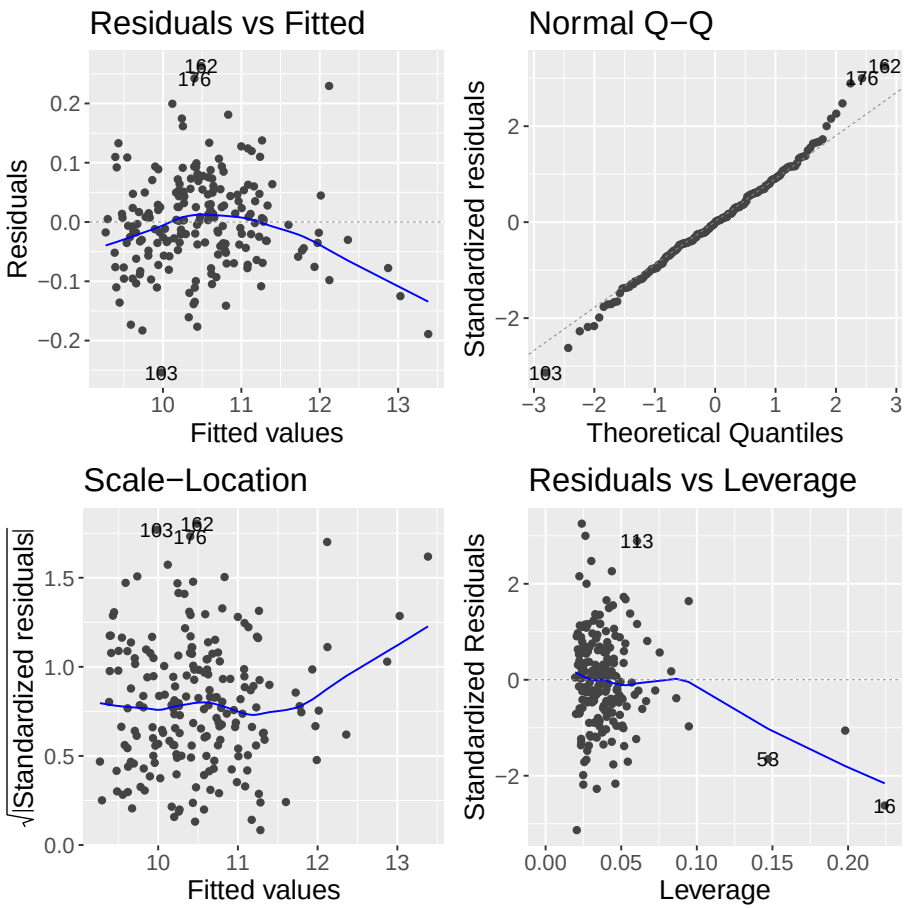
Remedies: this is a harder situation. When only few interactions are considered, a possible strategy would be to include all of them and perform backward selection. But then the p-values returned by `lm` for the parameters related to the interactions (here `Gender`, `Age`, and `Gender:Age`) will often be incorrect. Otherwise adding all possible interactions and use a lasso/group-lasso penalty with post selection inference. If the aim is prediction (and no inference), use more flexible but less interpretable models (e.g. random forest).

- A last possible issue arise when the **outcome variable is not studied on the right scale**. Consider the model using a square root transformation:

```
e.sqrt.lm <- lm(sqrt(Y1) ~ Gender*Age + Gene + BMI + I(BMI^2), data = dfW)
```

Diagnostic plots indicates lack of fit (first line, first plot) and heteroschedasticity (second line first plot):

```
autoplot(e.sqrt.lm)
```



We can use cumres and see that the link function seems inappropriate:

```
cumres(e.sqrt.lm, variable = "predicted")
```

```
          p-value(Sup) p-value(L2)
predicted            0       0.001

Based on 1000 realizations.
```

In that case a box-cox transformation can be useful as it suggests to square the outcome:

```
M <- MASS::boxcox(e.sqrt.lm, lambda = seq(-1,4,by=0.1))
M$x[which.max(M$y)]
```

```
[1] 1.828283
```

Note that it seems to sometimes also suggest weird transformations:

```
M <- MASS::boxcox(lm(log(Y1) ~ Gender*Age + Gene + BMI + I(BMI^2), data =
    dfW), lambda = seq(-10,10,by=0.1))
M$x[which.max(M$y)]
```

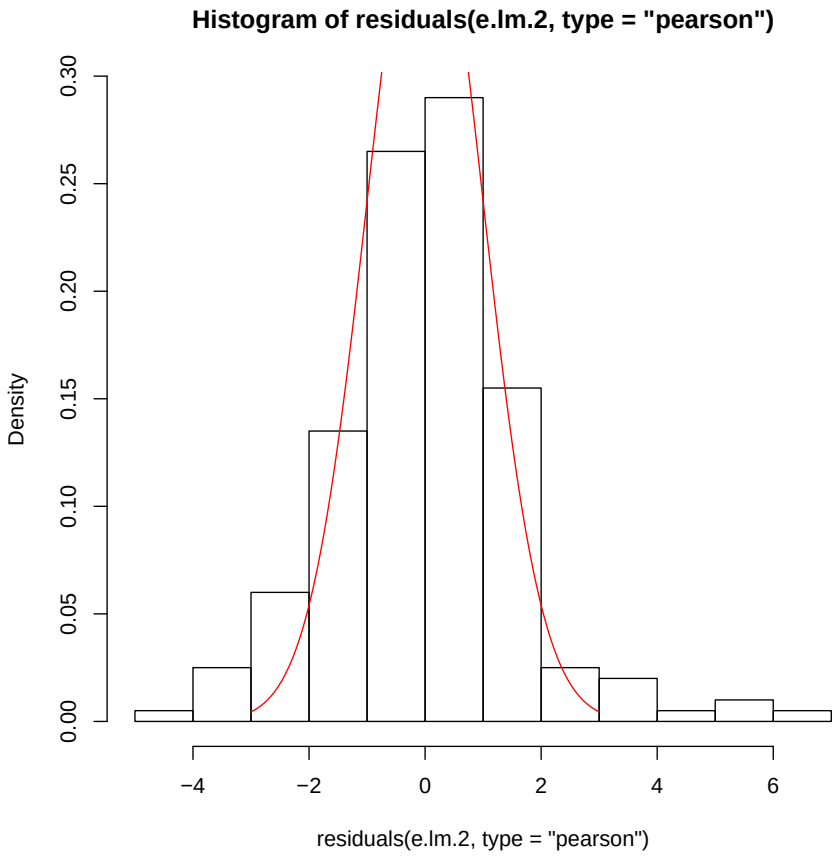
```
[1] 5.4
```

(the results should be 0)

4.4.3 (A4): normal distribution

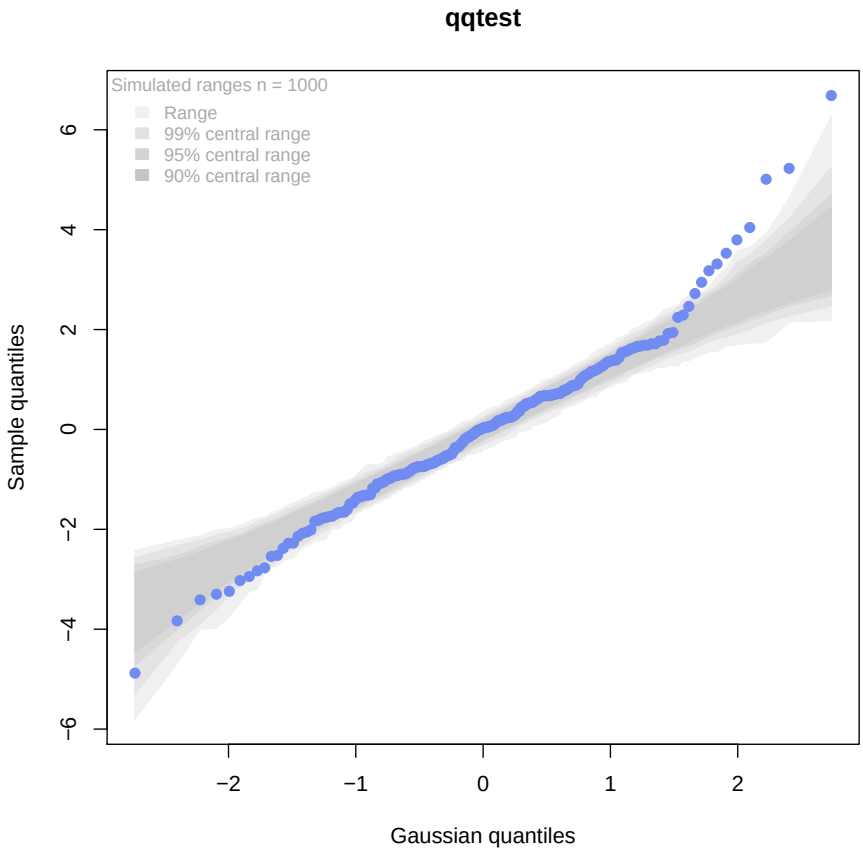
(A4) can be tested using an histogram of the standardized residuals:

```
hist(residuals(e.lm.2, type = "pearson"), freq = FALSE, breaks = 10)
curve(dnorm,-3,3,add =TRUE,col = "red")
```



where the histogram should be close to the shape of the standard normal distribution (red curve). We could reject **(A4)** but accept **(A4-bis)** in the case where the distribution has heavy tails but is still unimodal and symmetric. While intuitive, this method is sensitive to the discretization of the residuals values (argument break) and a qq-plot is often preferred:

```
qqtest::qqtest(residuals(e.lm.2, type = "pearson"))
```



Here the points should follow a straight line and be within the shaded area. We could reject **(A4)** but accept **(A4-bis)** in the case where deviation to the straight line mostly arise in the tails. Statistical test (like a shapiro test) are not recommended since they do not enable us to know whether we reject **(A4)** or **(A4bis)**.

Remedies: when **(A4)** is rejected but not **(A4-bis)**, the main concern is about the validity of the traditional asymptotic results. This is not critical in a linear regression where our variance estimator is consistent and the central limit theorem ensures asymptotic normality: instead of having exact p-values/CI they are only asymptotically valid. If the sample size is not too small they will hold; otherwise permutation test are a good alternative. In more complex models, robust standard errors or non-parametric bootstrap can be used for large enough samples to obtain

p-values/CI robust to deviation to the normal distribution.
A more serious problem arises when **(A4-bis)** is rejected. In that case one should consider whether the expected outcome is really relevant. Alternative approaches include transformation of the outcome or use of alternative regression models (quantile regression, probability index models, finite mixture models).

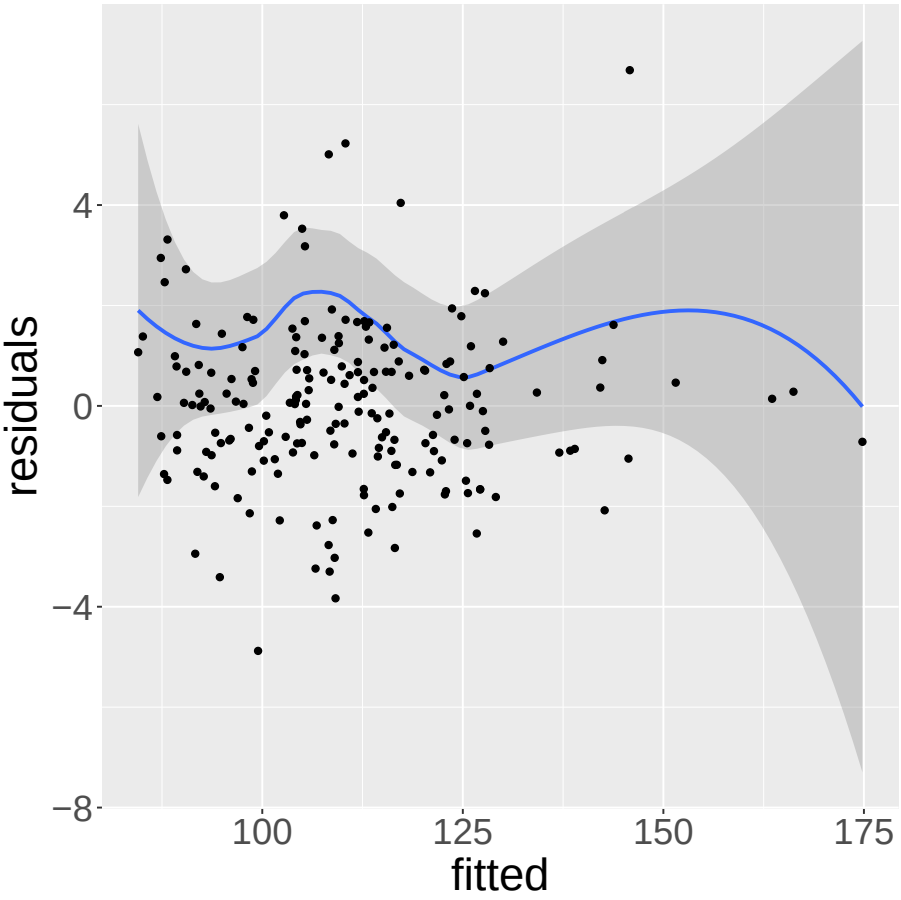
Note 1: the `type` argument indicates the type of residuals we want to extract. Raw residuals are $\hat{\varepsilon} = Y - \hat{Y}$, i.e. the observed minus the fitted values. In models more complex than a univariate linear regression, the raw residuals may not be iid. This makes it difficult to assess the validity of the assumptions. In such cases we display instead diagnostics for normalized residuals that, if the assumptions of the model are correct, should follow a standard normal distribution.

Note 2: an alternative to the `qqtest` function is the `qqPlot` function from the car package.

4.4.4 (A2): Homeschedasticity

Homoschedasticity can be inspected by displaying the residuals along the fitted values:

```
d$residuals <- residuals(e.lm.2, type = "pearson")
d$fitted <- fitted(e.lm.2)
gg <- ggplot(d, aes(x = fitted)) + ylab("residuals")
gg <- gg + geom_smooth(aes(y = residuals^2-1))
gg <- gg + geom_point(aes(y = residuals))
gg
```



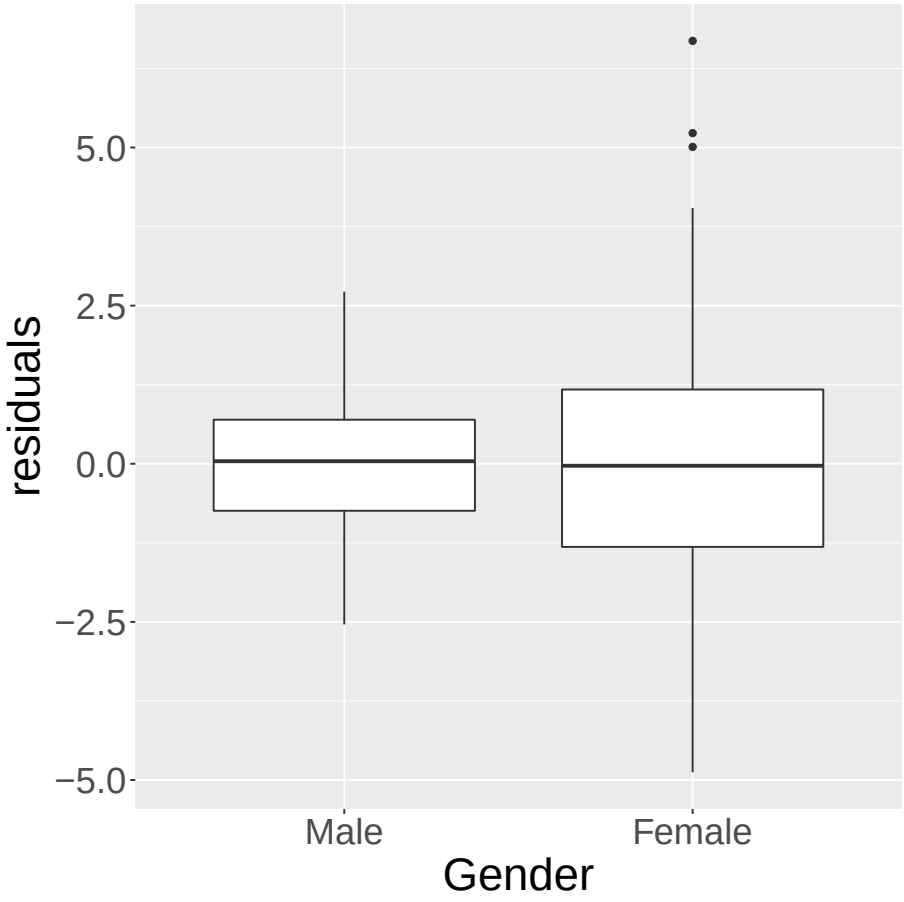
(see also the function `spreadLevelPlot` from the car package). It is also possible to have a global statistical test (Breusch-Pagan test):

```
ncvTest(e.lm.2)
```

```
Non-constant Variance Score Test
Variance formula: ~ fitted.values
Chisquare = 0.2765166, Df = 1, p = 0.59899
```

Alternatively one can look along a specific regressor:

```
gg <- ggplot(d, aes(x = Gender, y = residuals)) + ylab("residuals")
gg <- gg + geom_boxplot()
gg
```



or investigate look how the squared residuals relates to the regressors:

```
summary(lm(residuals(e.lm.2)^2 ~ Gender + Age + Gene + BMI, data = dfW))$
    coef
```

```
               Estimate Std. Error    t value     Pr(>|t|)
(Intercept)  3.27861402 3.07851696  1.0649979 2.882006e-01
GenderMale  -2.92345176 0.69766213 -4.1903546 4.227552e-05
Age         -0.03880792 0.03964689 -0.9788388 3.288787e-01
GeneB        0.41620970 0.87214618  0.4772247 6.337393e-01
GeneC        0.18989247 0.82390876  0.2304775 8.179636e-01
BMI          0.07868374 0.11627415  0.6767088 4.993968e-01
```

Remedies: in presence of global heteroschadasticity (first graph), transforming the outcome can be a solution. Otherwise one should reflect about possible source of heteroschadasticity (e.g. correlated observations, mixture of populations) and model them. When the heteroschadasticity is related to a single variable, one can for instance use the `gls` function to model this variance:

```
e.gls <- gls(Y1 ~ Gender + Age + Gene + BMI + I(BMI^2) + Gender:Age,
      data = dfW,
      weight = varIdent(form=~1|Gender))
summary(e.gls$modelStruct)
```

```
Variance function:
 Structure: Different standard deviations per stratum
 Formula: ~1 | Gender
 Parameter estimates:
    Male   Female
1.000000 2.096499
```

```
summary(
    lm(residuals(e.gls, type = "normalized")^2 ~ Gender + Age + Gene + BMI
    , data = dfW)
)$coef
```

```
               Estimate Std. Error    t value  Pr(>|t|)
(Intercept)  0.95513614 0.92435359  1.0333017 0.3027491
GenderMale   0.01093174 0.20947960  0.0521852 0.9584348
Age         -0.01802386 0.01190435 -1.5140566 0.1316390
GeneB        0.10417273 0.26187007  0.3978031 0.6912127
GeneC       -0.05737372 0.24738633 -0.2319195 0.8168450
BMI          0.02549938 0.03491241  0.7303817 0.4660380
```

4.4.5 (A5): Influential observations

The `influence` method can be used to output what is the impact of each observation on each estimated parameter:

```
if.lme <- influence(e.lm.2)
if.lme$coefficient[1:6,1:4]
```

```
    (Intercept)    GenderMale           Age         GeneB
1 -0.0768500758 -1.729088e-02 -8.430716e-06 -6.556256e-03
2  0.0033441763  8.848004e-04 -1.583090e-06 -1.330066e-03
3 -1.4634794147 -8.233340e-02 -2.155624e-04  4.249566e-02
4  0.0759756521 -4.035183e-03  4.852756e-05 -8.238371e-04
5 -0.0427789759 -1.599394e-03  2.073558e-05 -8.720082e-03
6  0.0008298817  4.867202e-05  2.398257e-07 -3.293940e-06
```

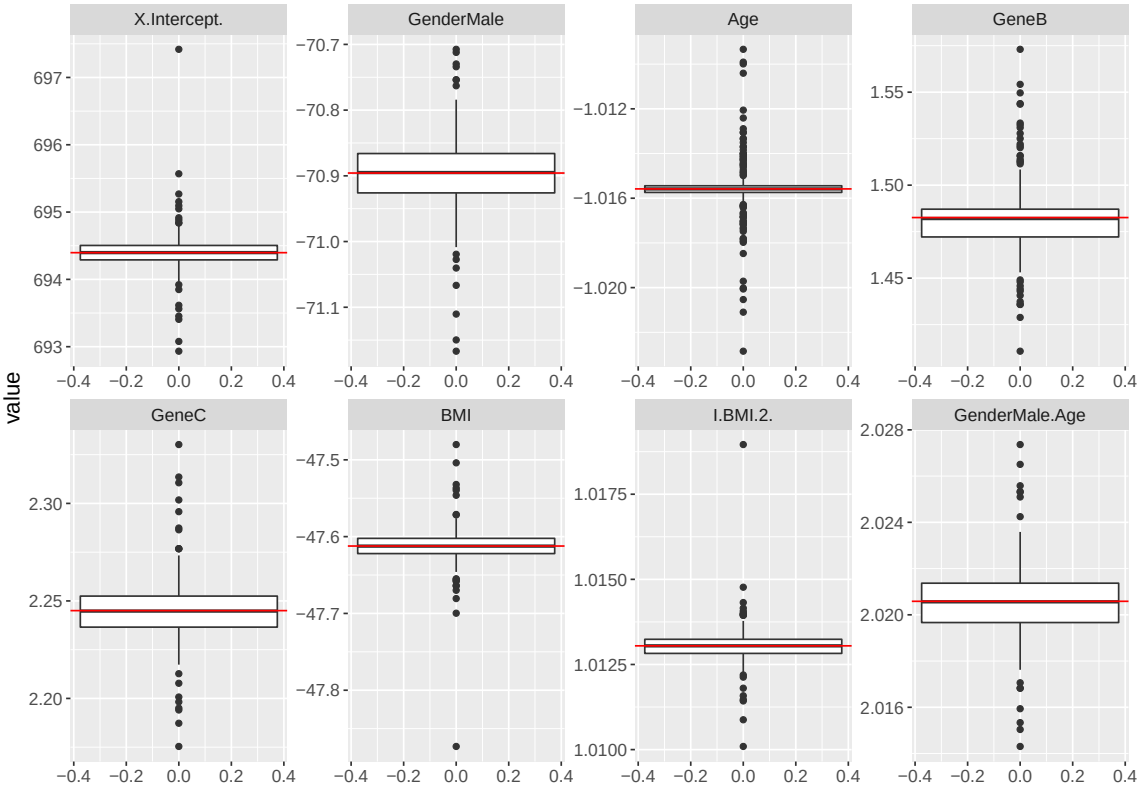
Here the value in the first line and third column indicates by how much is changed the Age effect when removing the first observation.

```
coef(update(e.lm.2, data = dfW[-1,]))-coef(e.lm.2)
```

```
 (Intercept)      GenderMale             Age           GeneB           GeneC             BMI
7.685008e-02   1.729088e-02   8.430716e-06   6.556256e-03   8.024692e-03  -7.151957e-03
    I(BMI^2) GenderMale:Age
1.540482e-04  -6.600643e-04
```

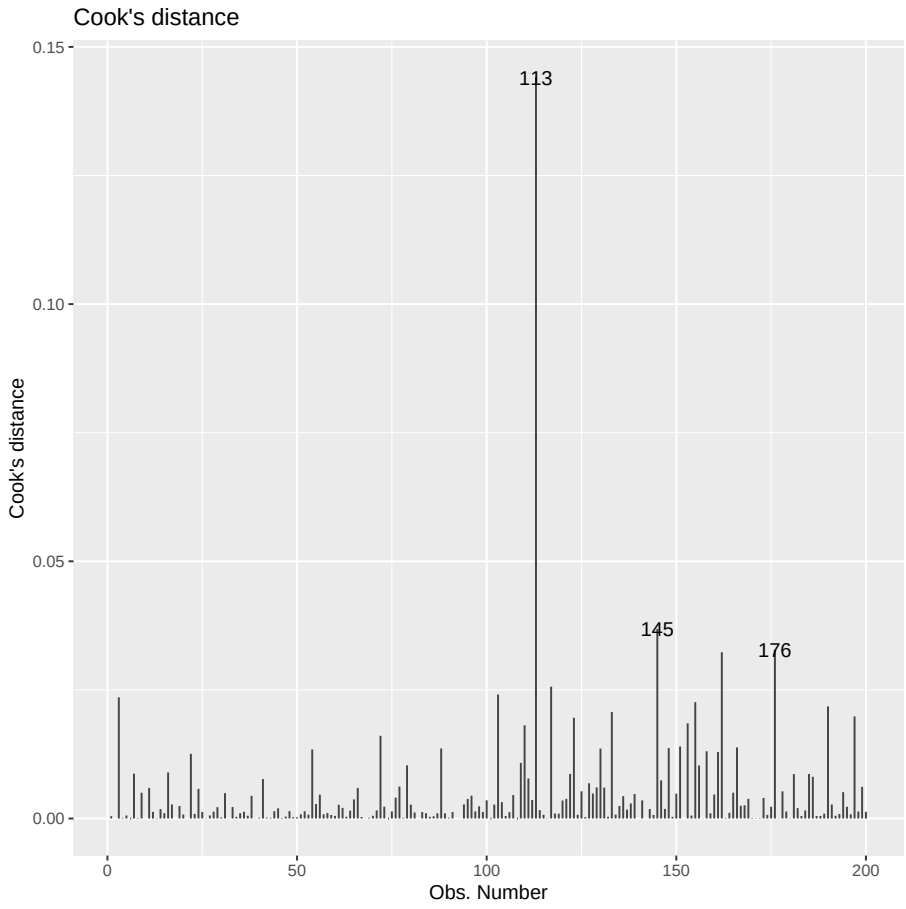
Large values (positive or negative) indicate influential observations. The following plot displaying in red the coefficient value and in black the influence of each individual can be useful:

```
df1.gg <- data.frame(id = "true", as.data.frame(t(coef(e.lm.2))))
df2.gg <- data.frame(id = as.character(1:NROW(d)),
       sweep(if.lme$coefficient, FUN = "+", MARGIN = 2, STATS = coef(e.lm
    .2)))
dfL1.gg <- reshape2::melt(df1.gg, id.vars = "id")
dfL2.gg <- reshape2::melt(df2.gg, id.vars = "id")
gg.inf <-  ggplot() + facet_wrap(~variable, scales = "free", nrow = 2)
gg.inf <- gg.inf + geom_boxplot(data = dfL2.gg, aes(y = value))
gg.inf <- gg.inf + geom_hline(data = dfL1.gg, aes(yintercept = value),
    color = "red")
gg.inf
```



When the aim is to perform prediction, global influence metrics such as Cook's distance can be useful:

```
autoplot(e.lm.2, which = 4)
```



4.4.6 Others [not recommanded unless specific reasons]

Some people recommand to check the correlation between the explanatory variables, with the argument that when very correlated it is difficult to disantangle effects and thus to interpret the regression coefficients. The VIF (variance inflation factor) is typically recommanded to check that with values higher than 5 considered as high:

```
car::vif(e.lm.2)
```

```
                GVIF Df GVIF^(1/(2*Df))
Gender      16.405278  1        4.050343
Age          2.107105  1        1.451587
Gene         1.070289  2        1.017127
BMI        153.493940  1       12.389267
I(BMI^2)   153.046502  1       12.371196
Gender:Age  17.937937  1        4.235320
```

I personnally don't recommand this as an automatic check since in many settings co-linearity can be better assessed from the meaning of the variables than from a statistical test. It is also quite unclear to me why 5 is a good cut-off and we see in this example that we get values close to five (or higher) even though there is no issue.

5 Partial residuals

5.1 With respect to one variable

The partial residuals with respect to age are defined by removing the effect of all the covariates but age on the outcome:

$$\hat{\varepsilon}_i^{Age} = Y_i - (\alpha + \beta_{Gender}\mathbb{1}_{Gender_i="Male"} + \beta_{GeneB}\mathbb{1}_{Gene_i="B"} + \beta_{GeneC}\mathbb{1}_{Gene_i="C"} + \beta_{BMI}BMI_i)$$

Using the following model coefficients:

```
coef(e.lm)
```

```
(Intercept)  GenderMale         Age       GeneB       GeneC         BMI
 91.5609817   3.7984283  -0.1358252   7.8328783   5.8120279   0.7364696
```

and considering the first individual:

```
d[1,]
```

```
  Id  Age Gender  BMI Gene    Y1    Y2       Y3 residuals   fitted
1  1 44.2   Male 23.8    A 109.1 120.8 131.7429 0.5188666 108.5811
```

the partial residual relative to age is:

$$\begin{aligned}\hat{\varepsilon}_1^{Age} &= 109.1 - (91.5610 + 3.7984 * 1 + 7.8329 * 0 + 5.8120 * 0 + 0.7365 * 23.8) \\ &= 109.1 - 112.8881 = -3.7881\end{aligned}$$

At the dataset level, this type of partial residual is centered around the expected value of the covariate times its effect (here $-0.1358252 * 34.4855 \approx -4.684$). These partial residuals can be computed using the `partialResidual` function from the butils package:

```
pRes.noI <- partialResiduals(e.lm, var = "Age", keep.intercept = FALSE)
head(pRes.noI)
```

```
   Id  Age Gender  BMI Gene    Y1    Y2       Y3     pFit ranef  pResiduals
1:  1 44.2   Male 23.8    A 109.1 120.8 131.7429 112.8874     0  -3.7873855
2:  2 41.3   Male 27.5    B 123.2 133.9 136.3850 123.4452     0  -0.2452012
3:  3 27.4   Male 30.6    A 140.6 154.0 136.1408 117.8954     0  22.7046216
4:  4 35.3   Male 25.2    C 104.4 116.2 125.0175 119.7305     0 -15.3304708
5:  5 37.1   Male 21.8    A 105.0 113.2 123.6257 111.4144     0  -6.4144463
6:  6 29.9   Male 18.1    C 125.9 136.2 131.7966 114.5015     0  11.3984631
```

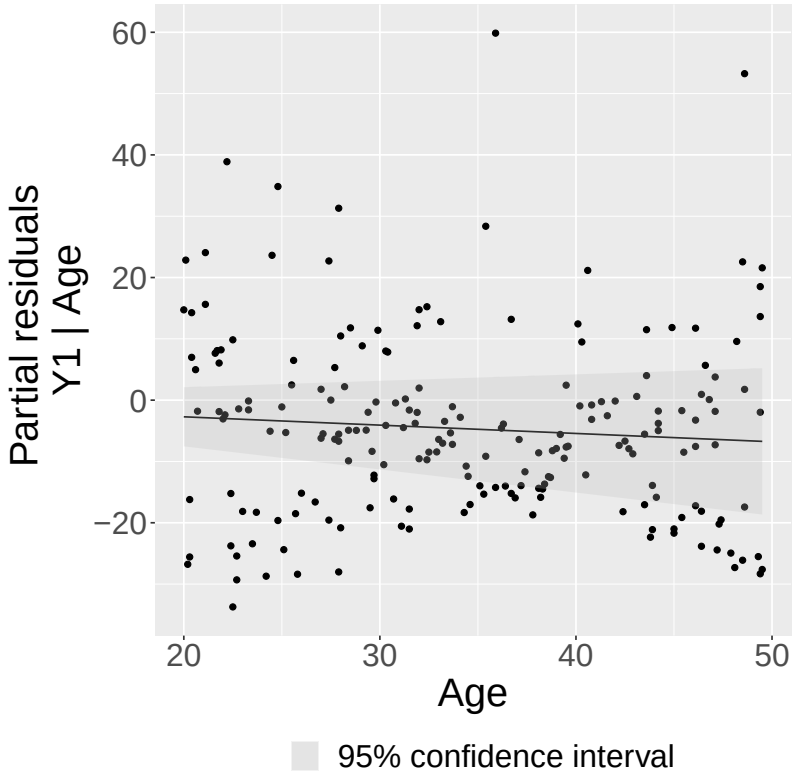
or manually:

```
keep.coef <- c("(Intercept)","GenderMale","GeneB","GeneC","BMI")
dfW$Y1[1] - model.matrix(e.lm)[1,keep.coef] %*% coef(e.lm)[keep.coef]
```

```
          [,1]
[1,] -3.787385
```

A graphical display can be obtained using the `autoplot` function (require the ggplot2 package):

```
gg <- autoplot(pRes.noI)
```



- An alternative definition do not remove the intercept effect:

$$\hat{\varepsilon}_i^{Age,\alpha} = Y_i - (\beta_{Gender}\mathbb{1}_{Gender_i="Male"} + \beta_{GeneB}\mathbb{1}_{Gene_i="B"} + \beta_{GeneC}\mathbb{1}_{Gene_i="C"} + \beta_{BMI}BMI_i)$$

so now the residuals are centered around the intercept plus the expected value of age times the age effect (here approximately 0).

As before the partial residuals can either be obtained via the `partialResiduals` function:

```
pRes.I <- partialResiduals(e.lm, var = "Age", keep.intercept = TRUE)
head(pRes.I)
```

```
   Id  Age Gender  BMI Gene    Y1    Y2       Y3     pFit ranef pResiduals
1:  1 44.2   Male 23.8    A 109.1 120.8 131.7429 21.32640     0   87.77360
2:  2 41.3   Male 27.5    B 123.2 133.9 136.3850 31.88422     0   91.31578
3:  3 27.4   Male 30.6    A 140.6 154.0 136.1408 26.33440     0  114.26560
4:  4 35.3   Male 25.2    C 104.4 116.2 125.0175 28.16949     0   76.23051
5:  5 37.1   Male 21.8    A 105.0 113.2 123.6257 19.85346     0   85.14654
6:  6 29.9   Male 18.1    C 125.9 136.2 131.7966 22.94056     0  102.95944
```

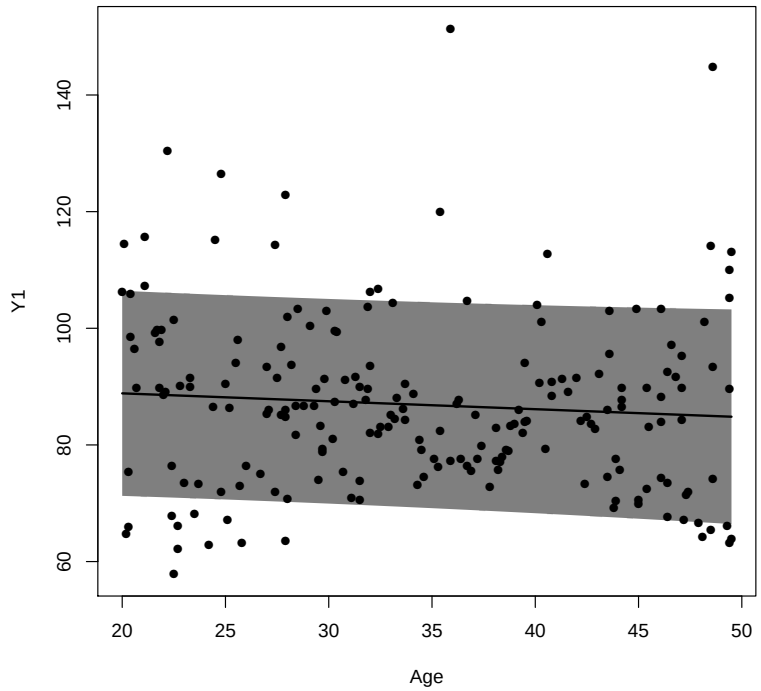
or manually:

```
keep.coef <- c("GenderMale","GeneB","GeneC","BMI")
dfW$Y1[1] - model.matrix(e.lm)[1,keep.coef] %*% coef(e.lm)[keep.coef]
```

```
        [,1]
[1,] 87.7736
```

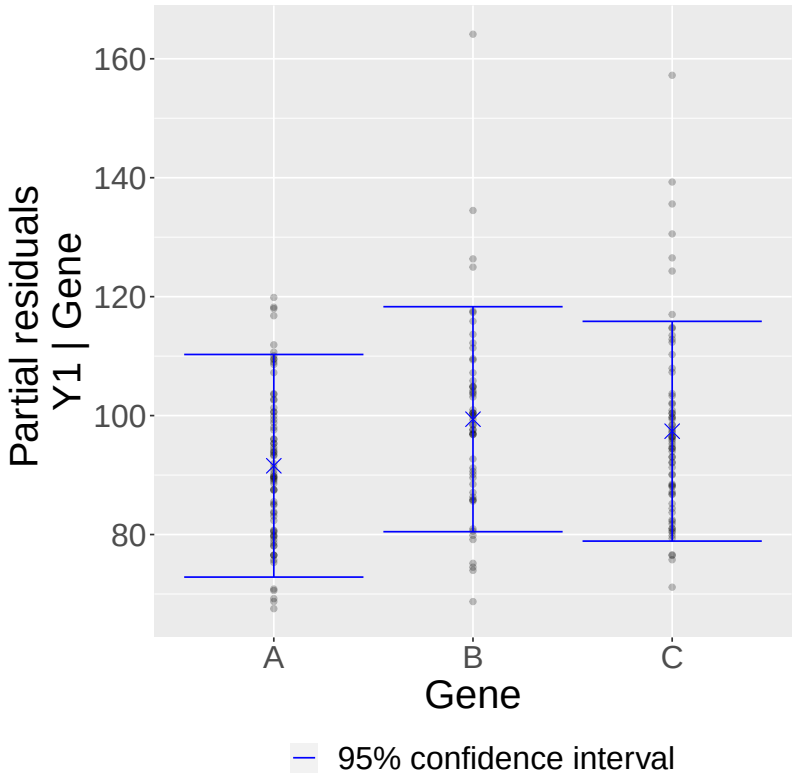
This corresponds to what the `plotConf` function is displaying (R package lava available on CRAN):

```
lava::plotConf(e.lm, var1 = "Age")
```



Note that it is also possible to display the partial residuals for a categorical variable:

```
pRes.cat <- partialResiduals(e.lm, var = "Gene", keep.intercept = TRUE)
gg <- autoplot(pRes.cat)
gg
```



5.2 With respect to an interaction between two variables (one continuous, one categorical)

Consider now a model where the age effect can be different for males and females:

```
e.lmI <- lm(Y1 ~ Gender * Age + Gene + BMI, data = dfW)
```

The partial residuals can be defined in a similar way as before. Here the effect of Age and Gender (and their interaction) are not substracted from the outcome:

```
gg <- autoplot(partialResiduals(e.lmI, var = c("Age","Gender")))
```

5.3 Customizing a partial residual plot

The autoplot function returns the ggplot object:

```
gg <- autoplot(partialResiduals(e.lm, var = "Gene", keep.intercept = TRUE)
    )
class(gg)
```

```
[1] "gg"     "ggplot"
```

So it can be easily customized, e.g. the text can be made bigger by doing:

```
gg + theme(text = element_text(size=25))
```