lineaRmodels

Léo Belzile

version of 2018-12-01

Contents

Preliminary remarks

This is a web complement to MATH 341 (Linear Models), a first regression course for EPFL mathematicians.

We shall use the **R** programming language througout the course (as it is free and it is used in other statistics courses at EPFL). Visit the R-project website[1] to download the program. The most popular graphical cross-platform front-end is RStudio Desktop[2].

R is an object-oriented interpreted language. It differs from usual programming languages in that it is designed for interactive analyses.

Since **R** is not a conventional programming language, my teaching approach will be learning-by-doing. The benefit of using *Rmarkdown* is that you see the output directly and you can also copy the code.

[1] https://cran.r-project.org/
[2] https://www.rstudio.com/products/rstudio/download/

Chapter 1

Introduction

You can find several introductions to **R** online. Have a look at the **R** manuals[1] or better at contributed manuals[2]. A nice official reference is An introduction to **R**[3]. You may wish to look up the following chapters of the **R** language definition (Evaluation of expressions[4] and part of the *Objects* chapter[5]).

If you favor online courses, Data Camp offers a free introduction to R[6].

1.1 Basics of R

1.1.1 Help

Help can be accessed via `help` or simply `?`. If you do not know what to query, use `??` in front of a string, delimited by captions `" "` as in `??"Cholesky decomposition"`. Help is your best friend if you don't know what a function does, what are its arguments, etc.

1.1.2 Basic commands

Basic **R** commands are fairly intuitive, especially if you want to use **R** as a calculator. Elementary functions such as `sum`, `min`, `max`, `sqrt`, `log`, `exp`, etc., are self-explanatory.

Some unconventional features of the language:

- Use `<-` to assign to a variable, and `=` for matching arguments inside functions
- Indexing in **R** starts at 1, **not** zero.
- Most functions in **R** are vectorized, so avoid loops as much as possible.
- Integers are obtained by appending `L` to the number, so `2L` is an integer and `2` a double.

Besides integers and doubles, the common types are - logicals (`TRUE` and `FALSE`); - null pointers (`NULL`), which can be assigned to arguments; - missing values, namely `NA` or `NaN`. These can also be obtained a result of invalid mathematical operations such as `log(-2)`.

The above illustrates a caveat of **R**: invalid calls will often returns *something* rather than an error. It is therefore good practice to check that the output is sensical.

[1] https://cran.r-project.org/manuals.html
[2] https://cran.r-project.org/other-docs.html
[3] http://colinfay.me/intro-to-r/index.html
[4] http://colinfay.me/r-language-definition/evaluation-of-expressions.html
[5] http://colinfay.me/r-language-definition/objects.html
[6] https://www.datacamp.com/courses/free-introduction-to-r

1.1.3 Linear algebra in R

R is an object oriented language, and the basic elements in **R** are (column) vector. Below is a glossary with some useful commands for performing basic manipulation of vectors and matrix operations:

- `c` as in _c_oncatenates creates a vector
- `cbind` (`rbind`) binds column (row) vectors
- `matrix` and `vector` are constructors
- `diag` creates a diagonal matrix (by default with ones)
- `t` is the function for transpose
- `solve` performs matrix inversion
- `%*%` is matrix multiplication, `*` is element-wise multiplication
- `crossprod(A, B)` calculates the cross-product $\mathbf{A}^\top\mathbf{B}$, `t(A) %*% B`, of two matrices `A` and `B`.
- `eigen/chol/qr/svd` perform respectively an eigendecomposition/Cholesky/QR/singular value decomposition of a matrix
- `rep` creates a vector of duplicates, `seq` a sequence. For integers i, j with $i < j$, `i:j` generates the sequence $i, i+1, \ldots, j-1, j$.

Subsetting is fairly intuitive and general; you can use vectors, logical statements. For example, if `x` is a vector, then

- `x[2]` returns the second element
- `x[-2]` returns all but the second element
- `x[1:5]` returns the first five elements
- `x[(length(x) - 5):length(x)]` returns the last five elements
- `x[c(1, 2, 4)]` returns the first, second and fourth element
- `x[x > 3]` return any element greater than 3. Possibly an empty vector of length zero!
- `x[ x < -2 | x > 2]` multiple logical conditions.
- `which(x == max(x))` index of elements satisfying a logical condition.

For a matrix `x`, subsetting now involves dimensions: `[1,2]` returns the element in the first row, second column. `x[,2]` will return all of the rows, but only the second column. For lists, you can use `[[` for subsetting by index or the `$` sign by names.

1.1.4 Packages

The great strength of **R** comes from its contributed libraries (called packages), which contain functions and datasets provided by third parties. Some of these (`base`, `stats`, `graphics`, etc.) are loaded by default whenever you open a session.

To install a package from CRAN, use `install.packages("package")`, replacing `package` by the package name. Once installed, packages can be loaded using `library(package)`; all the functions in `package` will be available in the environment.

There are drawbacks to loading packages: if an object with the same name from another package is already present in your environment, it will be hidden. Use the double-colon operator `::` to access a single object from an installed package (`package::object`).

1.2 Tutorial 1

1.2.1 Datasets

- datasets are typically stored inside a `data.frame`, a matrix-like object whose columns contain the variables and the rows the observation vectors.

- The columns can be of different types (`integer`, `double`, `logical`, `character`), but all the column vectors must be of the same length.
- Variable names can be displayed by using `names(faithful)`.
- Individual columns can be accessed using the column name using the `$` operator. For example, `faithful$eruptions` will return the first column of the `faithful` dataset.
- To load a dataset from an (installed) **R** package, use the command `data` with the name of the `package` as an argument (must be a string). The package `datasets` is loaded by default whenever you open **R**, so these are always in the search path.

The following functions can be useful to get a quick glimpse of the data:

- `summary` provides descriptive statistics for the variable.
- `str` provides the first few elements with each variable, along with the dimension
- `head` (`tail`) prints the first (last) n lines of the object to the console (default is $n = 6$).

We start by loading a dataset of the Old Faithful Geyser of Yellowstone National park and looking at its entries.

```
# Load Old faithful dataset
data(faithful, package = "datasets")
# Query the database for documentation
?faithful
# look at first entries
head(faithful)
```

```
##   eruptions waiting
## 1     3.600      79
## 2     1.800      54
## 3     3.333      74
## 4     2.283      62
## 5     4.533      85
## 6     2.883      55
```

```
str(faithful)
```

```
## 'data.frame':    272 obs. of  2 variables:
##  $ eruptions: num  3.6 1.8 3.33 2.28 4.53 ...
##  $ waiting  : num  79 54 74 62 85 55 88 85 51 85 ...
```

```
# What kind of object is faithful?
class(faithful)
```

```
## [1] "data.frame"
```

Other common classes of objects:

- `matrix`: an object with attributes `dim`, `ncol` and `nrow` in addition to `length`, which gives the total number of elements.
- `array`: a higher dimensional extension of `matrix` with arguments `dim` and `dimnames`.
- `list`: an unstructured class whose elements are accessed using double indexing `[[ ]]` and elements are typically accessed using `$` symbol with names. To delete an element from a list, assign `NULL` to it.
- `data.frame` is a special type of list where all the elements are vectors of potentially different type, but of the same length.

1.2.2 Graphics

The `faithful` dataset consists of two variables: the regressand `waiting` and the regressor `eruptions`. One could postulate that the waiting time between eruptions will be smaller if the eruption time is small, since pressure needs to build up for the eruption to happen. We can look at the data to see if there is a linear relationship between the variables.

An image is worth a thousand words and in statistics, visualization is crucial. Scatterplots are produced using the function `plot`. You can control the graphic console options using `par` — see `?plot` and `?par` for a description of the basic and advanced options available.

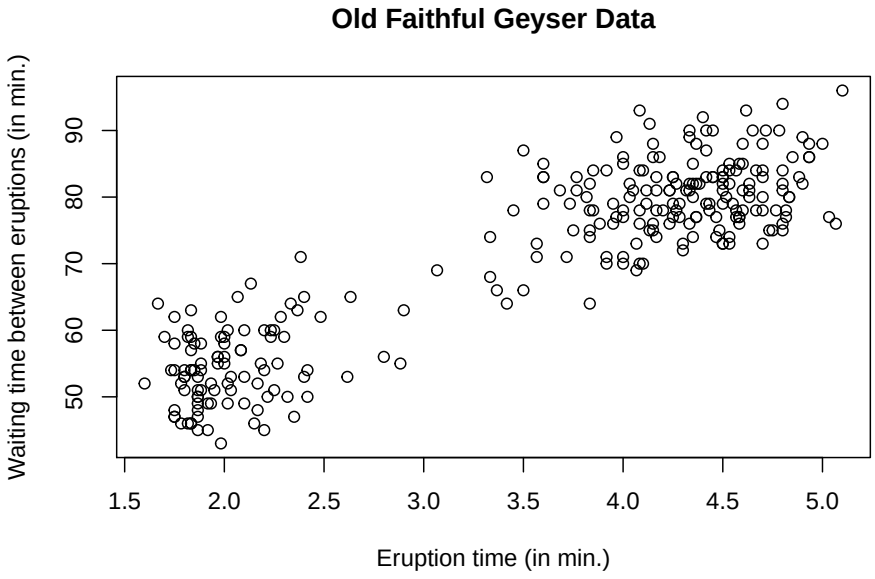
Once `plot` has been called, you can add additional observations as points (lines) to the graph using `point` (`lines`) in place of `plot`. If you want to add a line (horizontal, vertical, or with known intercept and slope), use the function `abline`.

Other functions worth mentioning at this stage:

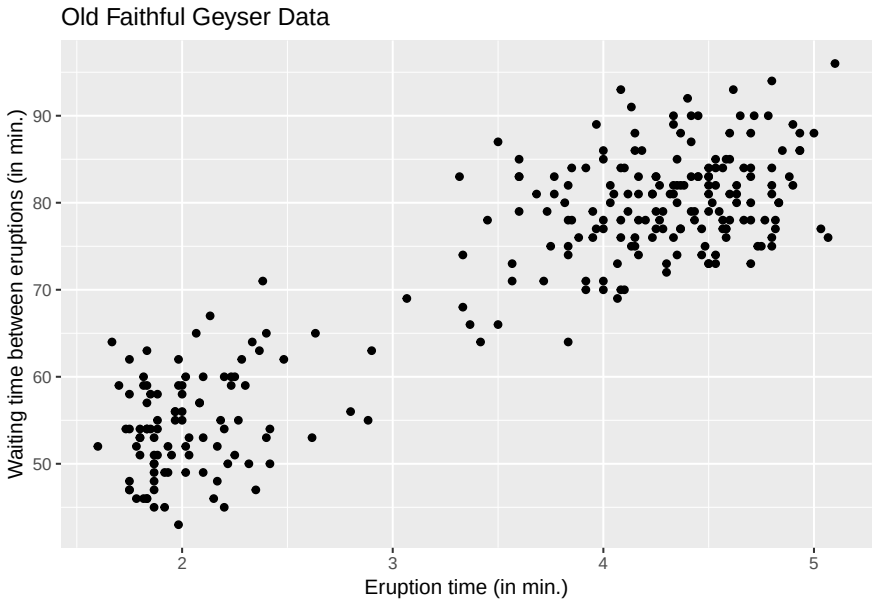
- `boxplot` creates a box-and-whiskers plot
- `hist` creates an histogram, either on frequency or probability scale (option `freq = FALSE`). `breaks` control the number of bins. `rug` adds lines below the graph indicating the value of the observations.
- `pairs` creates a matrix of scatterplots, akin to `plot` for data frame objects.

There are two options for basic graphics: the base graphics package and the package `ggplot2`. The latter is a more recent proposal that builds on a modular approach and is more easily customizable — I suggest you stick to either and `ggplot2` is a good option if you don't know **R** already, as the learning curve will be about the same. Even if the display from `ggplot2` is nicer, this is no excuse for not making proper graphics. Always label the axis and include measurement units!

```
# Scatterplots
# Using default R commands
plot(waiting ~ eruptions, data = faithful,
     xlab = "Eruption time (in min.)",
     ylab = "Waiting time between eruptions (in min.)",
     main = "Old Faithful Geyser Data")
```



```
#using the grammar of graphics (more modular)
#install.packages("ggplot2") #do this once only
library(ggplot2)
ggplot2::ggplot(data = faithful, aes(x = eruptions, y = waiting)) +
  geom_point() +
  labs(title = "Old Faithful Geyser Data",
       x = "Eruption time (in min.)",
       y = "Waiting time between eruptions (in min.)")
```



A simple linear model of the form

$$y_i = \beta_0 + \beta_1 x_i + \varepsilon_i,$$

where ε_i is a noise variable with expectation zero and $\mathbf{x}$ = eruptions and $\boldsymbol{y}$ = waiting. We first create a matrix with a column of $\mathbf{1}_n$ for the intercept. We bind vectors by column (`cbind`) into a matrix, recycling arguments if necessary. Use `$` to obtain a column of the data frame based on the name of the variable (partial matching is allowed, e.g., `faithful$er` is equivalent to `faithful$eruptions` in this case).

```
## Manipulating matrices
n <- nrow(faithful)
p <- ncol(faithful)
y <- faithful$waiting
X <- cbind(1, faithful$eruptions)
```

1.2.3 Projection matrices

Recall that $\mathbf{H_X} \equiv \mathbf{X}(\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top$ is the orthogonal projection matrix onto $\mathsf{span}(\mathbf{X})$. The latter has $p = 2$ eigenvalues equal to 1, is an $n \times n$ matrix of rank p, is symmetric and idempotent.

$\mathbf{H_X}$ is a great theoretical tool, but make no mistake: we will never use it in practice other than to verify statements made in class. The underlying reason is that it is an $n \times n$ matrix, so storage is costly if n is large. In practice, there are other ways to obtain quantities of interest such as coefficients, residuals and fitted values.

We can verify the properties of $\mathbf{H_X}$ numerically.

Whereas we will frequently use == to check for equality of booleans, the latter should be avoided for comparisons because computer arithmetic is exact only in base 2. For example, `1/10 + 2/10 - 3/10 == 0` will return `FALSE`, whereas `all.equal(1/10 + 2/10 - 3/10, 0)` will return `TRUE`. Use `all.equal` to check for equalities.

```
Hx <- X %*% solve(crossprod(X)) %*% t(X)
# Create projection matrix onto complement
# `diag(n)` is the n by n identity matrix
Mx <- diag(n) - Hx
#Check that projection leaves X invariant
isTRUE(all.equal(X, Hx %*% X))
```

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

```
#Check that orthogonal projection maps X to zero matrix of dimension (n, p)
isTRUE(all.equal(matrix(0, nrow = n, ncol = p), Mx %*% X))
```

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

```
#Check that the matrix Hx is idempotent
isTRUE(all.equal(Hx %*% Hx, Hx))
```

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

```
#Check that the matrix Hx is symmetric
isTRUE(all.equal(t(Hx), Hx))
```

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

```
#Check that only a two eigenvalue are 1 and the rest are zero
isTRUE(all.equal(eigen(Hx, only.values = TRUE)$values, c(rep(1, p), rep(0, n - p))))
```

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

```
#Check that the matrix has rank p
isTRUE(all.equal(Matrix::rankMatrix(Hx), p, check.attributes = FALSE))
```

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

Be careful: if `A` is an $n \times p$ matrix, `length(A)` returns the number of elements in the matrix, meaning np. Use `nrow(A)` for the number of observations.

1.3 Exercises

1.3.1 Auto dataset

- Install the **R** package `ISLR` and load the dataset `Auto`. Be careful, as **R** is case-sensitive.

- Query the help file for information about the dataset.
- Look at the first lines of `Auto`
- Create an explanatory variable `x` with horsepower and mileage per gallon as response `y`.
- Create a scatterplot of `y` against `x`. Is there evidence of a linear relationship between the two variables?
- Append a column vector of ones to `x` and create a projection matrix.
- Check that the resulting projection matrix is symmetric and idempotent.

1.4 Solutions

1.4.1 Exercise 1.4 - Oblique projections

Suppose that $\text{span}(\mathbf{X}) \neq \text{span}(\mathbf{W})$, that both $\mathbf{X}$ and $\mathbf{W}$ are full-rank $n \times p$ matrices such that $\mathbf{X}^\top\mathbf{W}$ and $\mathbf{W}^\top\mathbf{X}$ are invertible. An oblique projection matrix is of the form $\mathbf{P} \equiv \mathbf{X}(\mathbf{W}^\top\mathbf{X})^{-1}\mathbf{W}^\top$ and appears in instrumental variable regression. The oblique projection is such that $\text{im}(\mathbf{P}) = \text{span}(\mathbf{X})$, but $\text{im}(\mathbf{I}-\mathbf{P}) = \text{span}(\mathbf{W}^\perp)$. This fact is illustrated below.

We consider two non-parallel vectors in $\mathbb{R}^2$, $\mathbf{X}$ and $\mathbf{W}$.

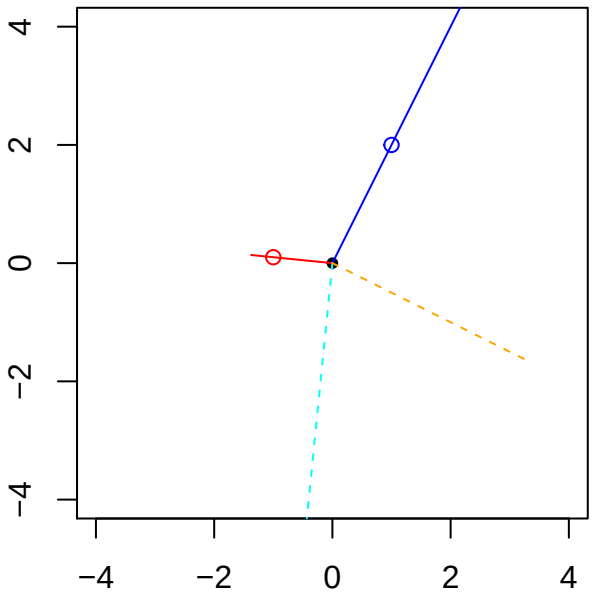
```
#Create two vectors (non-parallel)
x <- c(1, 2)
w <- c(-1, 0.1)
#Create oblique projection matrix
P <- x %*% solve(t(w) %*% x) %*% t(w)

isTRUE(all.equal((P %*% P), P)) #P is idempotent
```

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

```
P - t(P) #but not symmetric
```

```
##       [,1]   [,2]
## [1,] 0.000 -2.625
## [2,] 2.625  0.000
```

The figure below shows the projection of a third vector $\mathbf{v}$ (non-parallel to $\mathbf{X}$ or $\mathbf{W}$) onto the span of $\mathbf{P}$ (blue), $\mathbf{P}^\top$ (red), $\mathbf{I}_2-\mathbf{P}$ (dashed cyan) and $\mathbf{I}_2-\mathbf{P}^\top$ (dashed orange). The circles indicate the vectors $\mathbf{W}$ (red) and $\mathbf{X}$ (blue) on the plane. Notice that $\mathbf{I}_2-\mathbf{P}^\top \perp \mathbf{P}$, whereas $\mathbf{I}_2-\mathbf{P} \perp \mathbf{P}^\top$.



1.5 Summary of week 1

Let $\mathbf{X}$ be an $n \times p$ full-rank matrix ($p < n$). An $n \times n$ orthogonal projection matrix $\mathbf{H}$

- projects on to $\mathcal{V} \subseteq \mathbb{R}^n$, meaning $\mathbf{Hv} \in \mathcal{V}$ for any $\mathbf{v} \in \mathbb{R}^n$;
- is idempotent, meaning $\mathbf{H} = \mathbf{HH}$;
- is symmetric, meaning $\mathbf{H} = \mathbf{H}^\top$.

The projection matrix $\mathbf{H}$ is unique; if $\mathcal{V} = \mathcal{S}(\mathbf{X})$, then

$$\mathbf{H_X} = \mathbf{X}(\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top.$$

Since $\mathbf{X} : \mathbb{R}^n \to \mathbb{R}^p$, $\mathbf{H_X}$ has rank p. The orthogonal complement $\mathbf{M_X} \equiv \mathbf{I}_n - \mathbf{H_X}$ projects onto $\mathcal{S}^\perp(\mathbf{X})$.

Chapter 2

Computational considerations

In this tutorial, we will explore some basic **R** commands and illustrate their use on the Auto dataset (`Auto`) from the `ISLR` package.

2.1 Calculation of least square estimates

Consider as usual $\boldsymbol{y}$ and n-vector of response variables and a full-rank $n \times p$ design matrix $\mathbf{X}$. We are interested in finding the ordinary least square coefficient $\hat{\boldsymbol{\beta}}$, the fitted values $\hat{\boldsymbol{y}} = \mathbf{X}\hat{\boldsymbol{\beta}}$ and the residuals $\boldsymbol{e} = \boldsymbol{y} - \mathbf{X}\boldsymbol{\beta}$.

Whereas orthogonal projection matrices are useful for theoretical derivations, they are not used for computations. Building $\mathbf{H}_{\mathbf{X}}$ involves a matrix inversion and the storage of an $n \times n$ matrix. In Exercise series 2, we looked at two matrix decompositions: a singular value decomposition (SVD) and a QR decomposition. These are more numerically stable than using the normal equations $(\mathbf{X}^\top\mathbf{X})\boldsymbol{\beta} = \mathbf{X}^\top\boldsymbol{y}$ (the condition number of the matrix $\mathbf{X}^\top\mathbf{X}$ is the square of that of $\mathbf{X}$ — more on this later). The code related to the SVD and QR decompositions is provided for reference, so you can validate the derivations in the exercise. You won't need it in practice.

Optional material: for more details about the complexity and algorithms underlying the different methods, the reader is referred to these notes of Lee[1].

We can fit a simple linear model with an intercept and a linear effect for the weight,

$$\texttt{mpg}_i = \beta_0 + \texttt{hp}_i\beta_1 + \varepsilon_i.$$

We form the design matrix $(\mathbf{1}_n^\top, \texttt{hp}^\top)^\top$ and the vector of regressand `mpg`, then proceed with calculating the OLS coefficients $\hat{\boldsymbol{\beta}}$, the fitted values $\hat{\boldsymbol{y}}$ and the residuals $\boldsymbol{e}$.

We can compute first the ordinary least square estimates using the formula $\hat{\boldsymbol{\beta}} = (\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top\boldsymbol{y}$. The fitted values are $\hat{\boldsymbol{y}} = \mathbf{X}\hat{\boldsymbol{\beta}}$ and the residuals $\boldsymbol{e} = \boldsymbol{y} - \hat{\boldsymbol{y}}$.

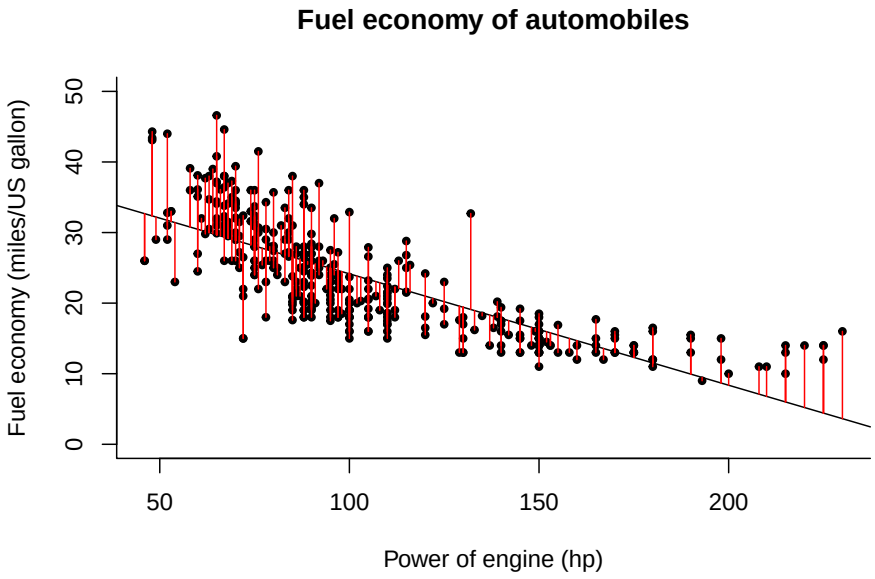
```
data(Auto, package = "ISLR")
y <- Auto$mpg
X <- cbind(1, Auto$horsepower)
n <- nrow(X)
p <- ncol(X)
# Estimation of beta_hat:
XtX <- crossprod(X)
Xty <- crossprod(X, y)
```

[1] www.math.uchicago.edu/~may/REU2012/REUPapers/Lee.pdf

```
# Solve normal equations
beta_hat <- as.vector(solve(XtX, Xty))
#same as beta_hat <- solve(t(X) %*% X) %*% t(X) %*% y

##Create residuals and fitted values
fitted <- as.vector(X %*% beta_hat)
res <- y - fitted
```

The residuals $\boldsymbol{e} = \boldsymbol{y} - \hat{\boldsymbol{y}}$ can be interpreted as the *vertical* distance between the regression slope and the observation. For each observation y_i, a vertical line at distance e_i is drawn from the prediction $\hat{y}_i$.

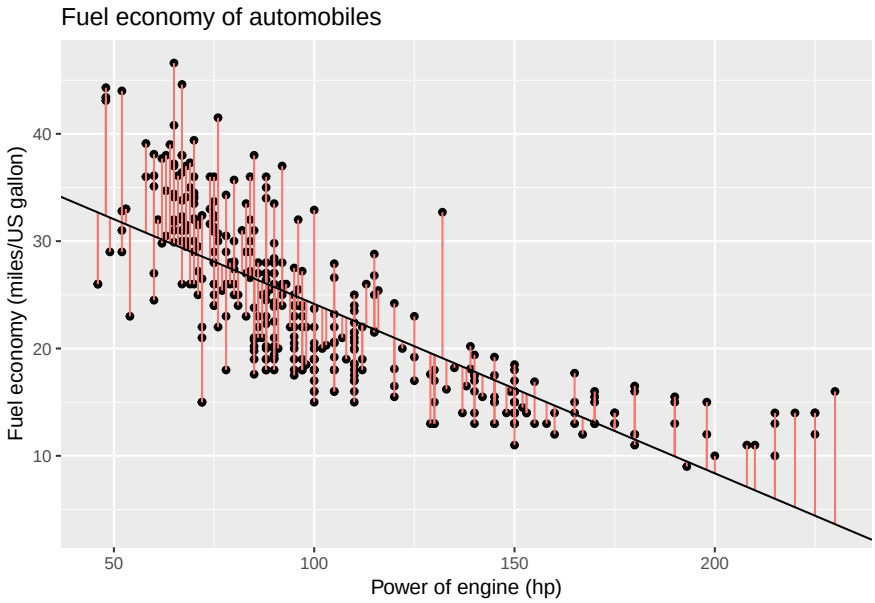
```
plot(mpg ~ horsepower,  data = Auto,
     xlab = "Power of engine (hp)",
     ylab = "Fuel economy (miles/US gallon)",
     main = "Fuel economy of automobiles",
     ylim = c(0, 50),
     # the subsequent commands for `plot`  tweak the display
     # check for yourself the effect of removing them
     # bty = "l" gives L shaped graphical windows (not boxed)
     # pch = 20 gives full dots rather than empty circles for points
     bty = "l", pch = 20)
#Line of best linear fit
abline(a = beta_hat[1], b = beta_hat[2])

#Residuals are vertical distance from line to
for(i in 1:nrow(X)){
  segments(x0 = Auto$horsepower[i], y0 = fitted[i], y1 = fitted[i] + res[i], col = 2)
}
```



The same scatterplot, this time using `ggplot2`.

```
library(ggplot2, warn.conflicts = FALSE, quietly = TRUE)
#Create data frame with segments
vlines <- data.frame(x1 = Auto$horsepower, y1 = fitted, y2 = fitted + res)
```

```
ggg <- ggplot(Auto, aes(x = horsepower, y = mpg)) +
        geom_point() +
        labs(x = "Power of engine (hp)",
             y = "Fuel economy (miles/US gallon)",
             title = "Fuel economy of automobiles") +
      geom_segment(aes(x = x1, y = y1, xend = x1, yend = y2, color = "red"),
                   data = vlines, show.legend = FALSE) +
      geom_abline(slope = beta_hat[2], intercept = beta_hat[1])
print(ggg)
```



2.1.1 Interpretation of the coefficients

If the regression model is

$$y_i = \beta_0 + \mathrm{x}_{i1}\beta_1 + \mathrm{x}_{i2}\beta_2 + \varepsilon_i,$$

the interpretation of β_1 in the linear model is as follows: a unit increase in x leads to β_1 units increase in y, everything else (i.e., x_{i2}) being held constant.

For the `Auto` regression above, an increase of the power of the engine by one horsepower leads to an average decrease of 0.16 miles per US gallon in distance covered by the car. We could easily get an equivalent statement in terms of increase of the car fuel consumption for a given distance.

2.2 The `lm` function

The function `lm` is the workshorse for fitting linear models. It takes as input a formula: suppose you have a data frame containing columns `x` (a regressor) and `y` (the regressand); you can then call `lm(y ~ x)` to fit the linear model $y = \beta_0 + \beta_1 x + \varepsilon$. The explanatory variable y is on the left hand side, while the right hand side should contain the predictors, separated by a + sign if there are more than one. If you provide the data frame name using `data`, then the shorthand `y ~ .` fits all the columns of the data frame (but y) as regressors.

To fit higher order polynomials or transformations, use the `I` function to tell **R** to interpret the input "as is". Thus, `lm(y~x+I(x^2))`, would fit a linear model with design matrix $(\mathbf{1}_n, \mathbf{x}^\top, \mathbf{x}^2)^\top$. A constant is automatically included in the regression, but can be removed by writing `-1` or `+0` on the right hand side of the formula.

```
# The function lm and its output
fit <- lm(mpg ~ horsepower + I(horsepower^2), data = Auto)
fit_summary <- summary(fit)
```

The `lm` output will display OLS estimates along with standard errors, t values for the Wald test of the hypothesis $H_0 : \beta_i = 0$ and the associated P-values. Other statistics and information about the sample size, the degrees of freedom, etc., are given at the bottom of the table.

Many methods allow you to extract specific objects. For example, the functions `coef`, `resid`, `fitted`, `model.matrix` will return $\hat{\boldsymbol{\beta}}$, $\boldsymbol{e}$, $\hat{\boldsymbol{y}}$ and $\mathbf{X}$, respectively.

```
names(fit)
```

```
##  [1] "coefficients"  "residuals"     "effects"       "rank"
##  [5] "fitted.values" "assign"        "qr"            "df.residual"
##  [9] "xlevels"       "call"          "terms"         "model"
```

```
names(fit_summary)
```

```
##  [1] "call"          "terms"         "residuals"     "coefficients"
##  [5] "aliased"       "sigma"         "df"            "r.squared"
##  [9] "adj.r.squared" "fstatistic"    "cov.unscaled"
```

The following simply illustrates what has been derived in Exercise series 2. **R** has devoted functions that are coded more efficiently.

2.2.1 Singular value decomposition

The SVD decomposition in **R** returns a list with elements `u`, `d` and `v`. `u` is the orthonormal $n \times p$ matrix, `d` is a vector containing the diagonal elements of $\mathbf{D}$ and `v` is the $p \times p$ orthogonal matrix. Recall that the decomposition is

$$\mathbf{X} = \mathbf{U}\mathbf{D}\mathbf{V}^\top$$

and that $\mathbf{V}\mathbf{V}^\top = \mathbf{V}^\top\mathbf{V} = \mathbf{U}^\top\mathbf{U} = \mathbf{I}_p$. The matrix $\mathbf{D}$ contains the singular values of $\mathbf{X}$, and the diagonal elements d_{ii}^2 corresponds to the (ordered) eigenvalues of $\mathbf{X}^\top\mathbf{X}$.

The following shows how to use the SVD decomposition in **R**. This material is **optional** and provided for reference only.

```
svdX <- svd(X)
# Projection matrix
Hx <- tcrossprod(svdX$u)
# t(U) %*% U gives p by p identity matrix
all.equal(crossprod(svdX$u), diag(p))
```

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

```
# V is an orthogonal matrix
all.equal(tcrossprod(svdX$v), diag(p))
```

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

```
all.equal(crossprod(svdX$v), diag(p))
```

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

```
# D contains singular values
all.equal(svdX$d^2, eigen(XtX, only.values = TRUE)$values)
```

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

```
# OLS coefficient from SVD
beta_hat_svd <- c(svdX$v %*%  diag(1/svdX$d) %*% t(svdX$u) %*% y)
all.equal(beta_hat_svd, beta_hat)
```

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

2.2.2 QR decomposition

R uses a QR-decomposition to calculate the OLS estimates in the function `lm`. There are specific functions to return coefficients, fitted values and residuals. One can also obtain the $n \times p$ matrix $\mathbf{Q}_1$ and the upper triangular $p \times p$ matrix $\mathbf{R}$ from the thinned QR decomposition,

$$\mathbf{X} = \mathbf{Q}_1 \mathbf{R}.$$

The following shows how to use the QR decomposition in **R**. This material is **optional** and provided for reference only.

```
Hx <- X %*% solve(crossprod(X)) %*% t(X)
qrX <- qr(X)
Q1 <- qr.Q(qrX)
R <- qr.R(qrX)
# Compute beta_hat from QR
beta_hat_qr1 <- qr.coef(qrX, y) #using built-in function
beta_hat_qr2 <- c(backsolve(R, t(Q1) %*% y)) #manually
all.equal(beta_hat, beta_hat_qr1, check.attributes = FALSE)
```

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

```
all.equal(beta_hat, beta_hat_qr2, check.attributes = FALSE)
```

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

```
# Compute residuals
qre <- qr.resid(qrX, y)
all.equal(qre, c(y - X %*% beta_hat), check.attributes = FALSE)
```

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

```
# Compute fitted values
qryhat <- qr.fitted(qrX, y)
all.equal(qryhat, c(X %*% beta_hat), check.attributes = FALSE)
```

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

```
# Compute orthogonal projection matrix
qrHx <- tcrossprod(Q1)
all.equal(qrHx, Hx)
```

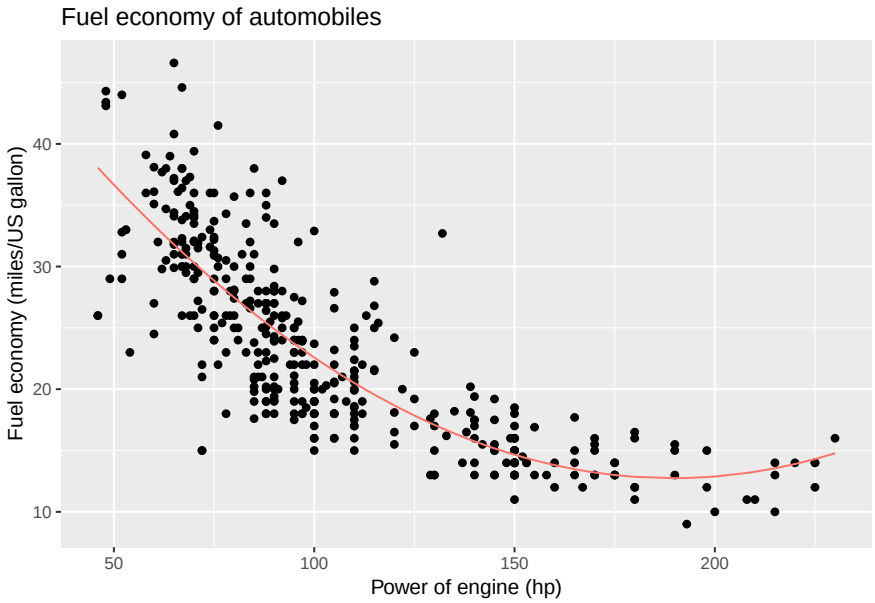
```
## [1] TRUE
```

2.3 The hyperplane of fitted values

In class, we presented a linear model for the `Auto` dataset of the form

$$\mathtt{mpg}_i = \beta_0 + \beta_1 \mathtt{hp}_i + \beta_2 \mathtt{hp}_i^2 + \varepsilon_i$$

and claimed this was a linear model. This is indeed true because we can form the design matrix $[\mathbf{1}_n, \mathtt{hp}, \mathtt{hp}^2]$ and obtain coefficients $\hat{\boldsymbol{\beta}}$. The graphical depiction is counterintuitive.



This quadratic curve is nothing like an hyperplane! Let $\boldsymbol{y} \equiv \mathtt{mpg}$, $\mathtt{x} = \mathtt{hp}$ and $\mathtt{z} = \mathtt{hp}^2$. But recall that we are working in three dimensions (the intercept gives the height of the hyperplane) and the coordinates of our hyperplane are

$$\beta_0 + \beta_1 x - y + \beta_2 z = 0.$$

However, the observations will always be such that $z = x^2$, so our fitted values will lie on a one-dimensional subspace of this hyperplane.

The following 3D depiction hopefully captures this better and shows the fitted hyperplane along with the line on which all the (x_i, z_i) observations lie.

2.4 (Centered) coefficient of determination

Recall the decomposition of observations into fitted and residual vectors,

$$\boldsymbol{y} = (\boldsymbol{y} - \mathbf{X}\hat{\boldsymbol{\beta}}) + \mathbf{X}\hat{\boldsymbol{\beta}} = \boldsymbol{e} + \hat{\boldsymbol{y}}$$

where $\boldsymbol{e} \equiv \mathbf{M}_{\mathbf{X}}\boldsymbol{y} \perp \hat{\boldsymbol{y}} \equiv \mathbf{H}_{\mathbf{X}}\boldsymbol{y}$.

The centered coefficient of determination, R_c^2 measures the proportion of variation explained by the centered fitted values relative to the centered observations, i.e.,

$$R_c^2 = \frac{\|\hat{\boldsymbol{y}} - \bar{y}\mathbf{1}_n\|^2}{\|\boldsymbol{y} - \bar{y}\mathbf{1}_n\|^2} = \frac{\|\hat{\boldsymbol{y}}\|^2 - \|\bar{y}\mathbf{1}_n\|^2}{\|\boldsymbol{y}\|^2 - \|\bar{y}\mathbf{1}_n\|^2}.$$

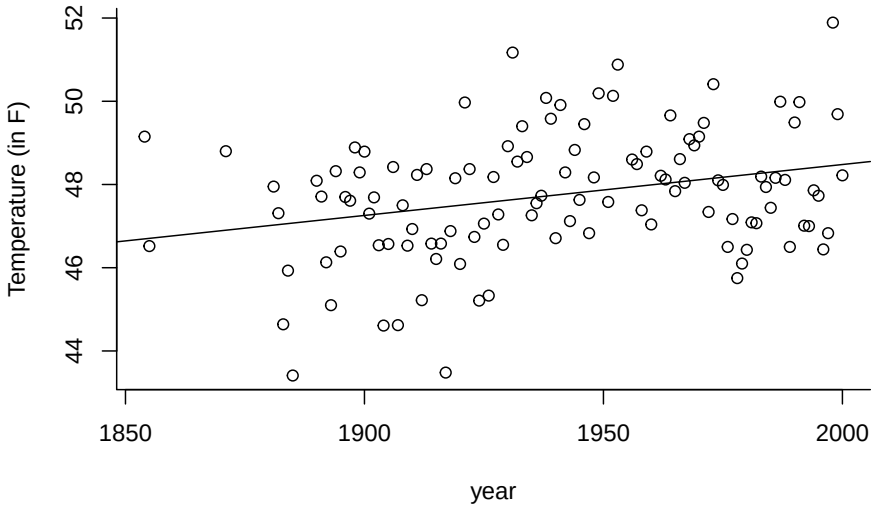
since the vectors $\bar{y}\mathbf{1}_n \perp \hat{\boldsymbol{y}} - \bar{y}\mathbf{1}_n$.

Provided that $\mathbf{1}_n \in \mathcal{S}(\mathbf{X})$, it is obvious that the fitted values $\hat{\boldsymbol{y}}$ are invariant to linear transformations of the covariates $\mathbf{X}$ (by which I mean you can transform the design matrix column by column, with $\mathbf{x}_i \mapsto \alpha_i + \mathbf{x}_i\gamma_i$ for $i = 1, \ldots, p$). Multiplicative changes in $\boldsymbol{y}$ lead to an equivalent change in $\boldsymbol{e}$ and $\hat{\boldsymbol{y}}$. However, location-changes in $\boldsymbol{y}$ are only reflected in $\hat{\boldsymbol{y}}$ (they are absorbed by the intercept). This is why R^2 is not invariant to location-changes in the response, since the ratio $\|\hat{\boldsymbol{y}}\|^2/\|\boldsymbol{y}\|^2$ increases to 1 if $\boldsymbol{y} \mapsto \boldsymbol{y} + a\mathbf{1}_n$.

This invariance is precisely the reason we dismissed R^2. For example, a change of units from Farenheit to Celcius, viz. $T_c = 5(T_F - 32)/9$, leads to different values of R^2:

```
data(aatemp, package = "faraway")
plot(temp ~ year, data = aatemp, ylab = "Temperature (in F)", bty = "l")
#Form design matrix and two response vectors
yF <- aatemp$temp
n <- length(yF)
yC <- 5/9*(aatemp$temp - 32)
```

```
X <- cbind(1, aatemp$year)
# Obtain OLS coefficients and fitted values
XtX <- solve(crossprod(X))
beta_hat_F <- XtX %*% crossprod(X, yF)
abline(a = beta_hat_F[1], b = beta_hat_F[2])
```




```
beta_hat_C <- XtX %*% crossprod(X, yC)
fitted_F <- X %*% beta_hat_F
fitted_C <- X %*% beta_hat_C
# Compute coefficient of determination
R2_F <- sum(fitted_F^2)/sum(yF^2)
R2_C <-  sum(fitted_C^2)/sum(yC^2)
#Centered R^2
R2c_F <- sum((fitted_F-mean(yF))^2)/sum((yF-mean(yF))^2)
R2c_C <-  sum((fitted_C-mean(yC))^2)/sum((yC-mean(yC))^2)
isTRUE(all.equal(R2c_F, R2c_C))
```

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

The difference $R^2(F) - R^2(C) = 0.00752$ is small because the R^2 value is very high, but the coefficient itself is also meaningless. In this example, $R^2(F) = 0.9991$, which seems to indicate excellent fit but in fact only 8.54% of the variability is explained by year and we do an equally good job by simply taking $\hat{y}_i = \bar{y}$.

R_c^2 makes the comparison between the adjusted linear model and the null model with only a constant, which predicts each $y_i (i = 1, \ldots, n)$ by the average $\bar{y}$.

If R_c^2 gives a very rough overview of how much explanatory power $\mathbf{X}$ has, it is not a panacea. If we add new covariates in $\mathbf{X}$, the value of R_c^2 necessarily increases. In the most extreme scenario, we could add a set of $n - p$ linearly independent vectors to $\mathbf{X}$ and form a new design matrix mX^* with those. The fitted values from running a regression with $\mathbf{X}^*$ will be exactly equal to the observations $\boldsymbol{y}$ and thus $R_c^2 = 1$. However, I hope it is clear that this model will *not* be useful. Overfitting leads to poor predictive performance; if we get a new set of $\mathbf{x}_*$, we would predict the unobserved y_* using its conditional average $\mathbf{x}_i^* \hat{\boldsymbol{\beta}}$ and this estimate will be rubish if we included too many meaningless covariates.

Other versions of R_c^2 exist that include a penalty term for the number of covariates; these are not widely used and can be negative in extreme cases. We will cover better goodness-of-fit diagnostics later in the course.

In **R**, the function `lm` returns R_c^2 by default (in the `summary` table, under the label `Multiple R-squared`. However, if you remove the intercept, you will get R^2 without warning! Contrast

```
mod <- lm(mpg ~ horsepower, data = Auto)
rsqc_lm <- summary(mod)$r.squared
#same model, now X = [1 horsepower] and y = mpg
X <- cbind(1, Auto$horsepower)
y <- Auto$mpg
rsq_lm <- summary(lm(y ~ X - 1))$r.squared

#Compute quantities manually
rsqc_man <- c(crossprod(fitted(mod) - mean(y)) / crossprod(y - mean(y)))
isTRUE(all.equal(rsqc_man, rsqc_lm))
```

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

```
rsq_man <- c(crossprod(fitted(mod))/crossprod(y))
isTRUE(all.equal(rsq_man, rsq_lm))
```

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

2.5 Summary of week 2

If $\mathbf{X}$ is an $n \times p$ design matrix containing *covariates* and $\boldsymbol{Y}$ is our response variable, we can obtain the *ordinary least squares* (OLS) coefficients for the linear model

$$\boldsymbol{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon}, \qquad \mathrm{E}(\boldsymbol{\varepsilon}) = \mathbf{0}_n,$$

by projecting $\boldsymbol{y}$ on to $\mathbf{X}$; it follows that

$$\mathbf{X}\hat{\boldsymbol{\beta}} = \mathbf{X}(\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top\boldsymbol{y}$$

and

$$\hat{\boldsymbol{\beta}} = (\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top\boldsymbol{y}.$$

The dual interpretation (which is used for graphical diagnostics), is the row geometry: each row corresponds to an individual and the response is a 1 dimensional point. $\hat{\boldsymbol{\beta}}$ describes the parameters of the hyperplane that minimizes the sum of squared Euclidean vertical distances between the fitted value $\hat{y}_i$ and the response y_i. The problem is best written using vector-matrix notation, so

$$\mathrm{argmin}_{\boldsymbol{\beta}} \sum_{i=1}^{n} (y_i - \mathbf{x}_i\boldsymbol{\beta})^2 \equiv \mathrm{argmin}_{\boldsymbol{\beta}} (\boldsymbol{y} - \mathbf{X}\boldsymbol{\beta})^\top (\boldsymbol{y} - \mathbf{X}\boldsymbol{\beta}) \equiv \boldsymbol{e}^\top\boldsymbol{e}.$$

The solution to the OLS problem has a dual interpretation in the column geometry, in which we treat the vector of stacked observations $(y_1, \ldots, y_n)^\top$ (respectively the vertical distances $(e_1, \ldots, e_n)^\top$) as elements of $\mathbb{R}^n$. There, the response $\boldsymbol{y}$ space can be decomposed into *fitted values* $\hat{\boldsymbol{y}} \equiv \mathbf{H}_{\mathbf{X}} = \mathbf{X}\hat{\boldsymbol{\beta}}$ and *residuals* $\boldsymbol{e} = \mathbf{M}_{\mathbf{X}} = \boldsymbol{y} - \mathbf{X}\hat{\boldsymbol{\beta}}$. By construction, $\boldsymbol{e} \perp \hat{\boldsymbol{y}}$.

We therefore get

$$\boldsymbol{y} = \hat{\boldsymbol{y}} + \boldsymbol{e}$$

and since these form a right-angled triangle, Pythagoras' theorem can be used to show that $\|\boldsymbol{y}\|^2 = \|\hat{\boldsymbol{y}}\|^2 + \|\boldsymbol{e}\|^2$.

2.6 Solutions

The following questions refer to the dataset `prostate` from the package `ElemStatLearn`.

a. Briefly describe the dataset.
b. Look at summaries of `lbph`. What likely value was imputed in places of zeros in ‘lbph} (before taking the logarithm)?
c. Produce a plot of the pair of variables `lcavol` and `lpsa` on the log and on the original scale. Comment on the relationship between `lcavol` and `lpsa`.
d. Fit a linear model using the log cancer volume as response variable, including a constant and the log prostate specific antigen as covariates. Obtain numerically the OLS estimates $\hat{\boldsymbol{\beta}}$ of the parameters, the fitted values $\hat{\boldsymbol{y}}$ and the residuals $\boldsymbol{e}$ using the formulas given in class.
e. Compare the quantities you obtained with the output of the function `lm`.
f. Add the fitted regression line to the scatterplot of `lcavol` against `lpsa`.
g. Interpret the changes in cancer volume (not the log cancer volume), including any units in your interpretations.
h. Obtain the orthogonal projection matrix $\mathbf{H}_{\mathbf{X}}$ and the OLS coefficients $\hat{\boldsymbol{\beta}}$ using a SVD decomposition of $\mathbf{X}$ (`svd`).
i. Compute the R_c^2 coefficient and compare with the one in `summary` output of the `lm` function. What can you say about the explanatory power of the covariate `lpsa`?

2.6.1 Exercise 3.5 - Prostate cancer

The following questions refer to the dataset `prostate` from the package `ElemStatLearn`.

a. Briefly describe the data set.

Running `?ElemStatLearn::prostate` gives the help file for the data set. Since we will be coming back to this example, detailed informations are provided below.

This data set was extracted from

> Stamey, T.A., Kabalin, J.N., McNeal, J.E., Johnstone, I.M., Freiha, F., Redwine, E.A. and Yang, N. (1989) Prostate specific antigen in the diagnosis and treatment of adenocarcinoma of the prostate: II. radical prostatectomy treated patients, Journal of Urology 141(5), 1076–1083.

This data set is described in Wakefield (2013), pp. 5-6.

> The data were collected on $n = 97$ men before radical prostatectomy, a major surgical operation that removes the entire prostate gland along with some surrounding tissue.
>
> In Stamey et al. (1989), prostate specific antigen (PSA) was proposed as a preoperative marker to predict the clinical stage of cancer. As well as modeling the stage of cancer as a function of PSA, the authors also examined PSA as a function of age and seven other histological and morphometric covariates.
>
> The BPH and capsular penetration variables originally contained zeros, and a small number was substituted before the log transform was taken. It is not clear from the original paper why the log transform was taken though PSA varies over a wide range, and so linearity of the mean model may be aided by the log transform. It is also not clear why the variable PGS45 was constructed.

The data set contains the following variables:

- `lcavol`: log of cancer volume, measured in milliliters (cc). The area of cancer was measured from digitized images and multiplied by a thickness to produce a volume.
- `lweight`: log of the prostate weight, measured in grams.
- `age`: The age of the patient, in years.
- `lbph`: log of the amount of benign prostatic hyperplasia (BPH), a noncancerous enlargement of the prostate gland, as an area in a digitized image and reported in cm^2.

- `svi`: seminal vesicle invasion, a 0/1 indicator of whether prostate cancer cells have invaded the seminal vesicle.
- `lcp`: log of the capsular penetration, which represents the level of extension of cancer into the capsule (the fibrous tissue which acts as an outer lining of the prostate gland), measured as the linear extent of penetration, in cm.
- `gleason`: Gleason score, a measure of the degree of aggressiveness of the tumor. The Gleason grading system assigns a grade (1–5) to each of the two largest areas of cancer in the tissue samples with 1 being the least aggressive and 5 the most aggressive; the two grades are then added together to produce the Gleason score.
- `pgg45`: percentage of Gleason scores that are 4 or 5.
- `lpsa`: log of prostate specific antigen (PSA), a concentration measured in ng/m

To load the data set, use

```
#Install package if you get an error message
#install.packages("ElemStatLearn")
data(prostate, package = "ElemStatLearn")
?ElemStatLearn::prostate
attach(prostate)
```

The command `attach` allows you to access column (variables) without using $ by adding the columns of the data frame to your work environment. **Always** detach the data once you are done with your analysis to avoid overriding or hidding variables.

b. Look at summaries of `lbph`. What likely value was imputed in places of zeros in `lbph` (before taking the logarithm)?

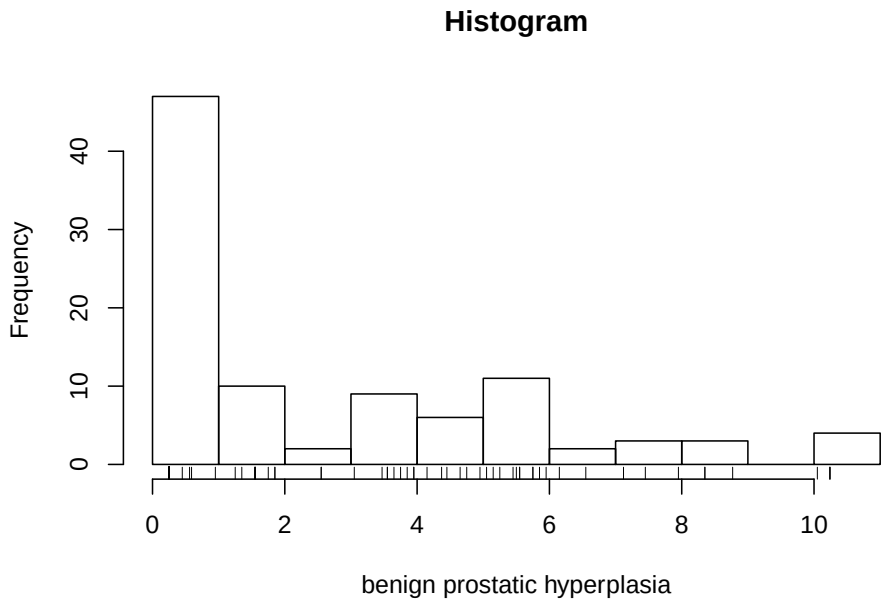
```
bph <- exp(lbph)
head(bph) #look up first elements
```

```
## [1] 0.25 0.25 0.25 0.25 0.25 0.25
```

```
min(bph) #return minimum
```

```
## [1] 0.25
```

```
hist(bph, main = "Histogram", xlab = "benign prostatic hyperplasia")
rug(bph)
```




```
#histogram, with lines below where the observations are
```

It seems likely that in order to take a logarithm, zeros were changed to 0.25. As such, we have to be careful with the interpretation of this coefficient if we include bph in the regression.

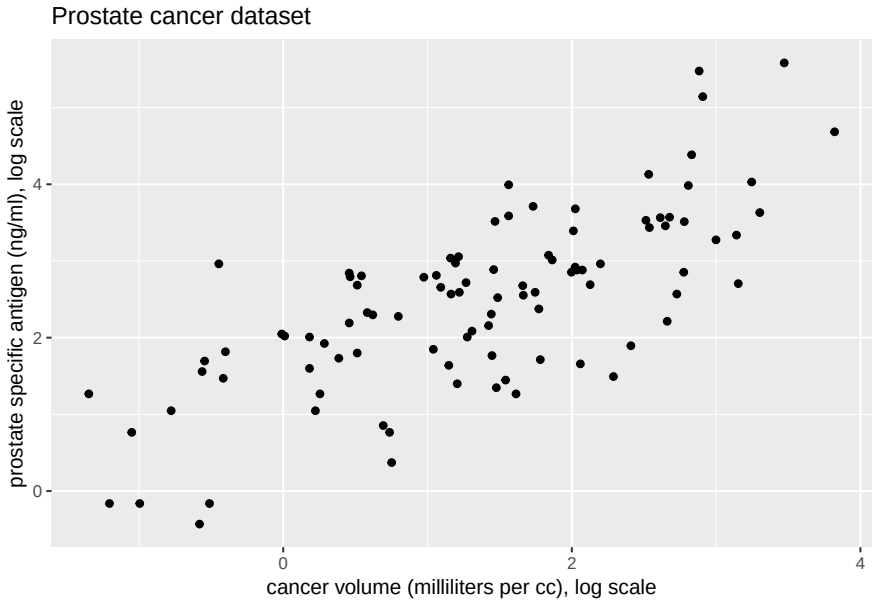
b. Produce a plot of the pair of variables `lcavol` and `lpsa` on the log and on the original scale. Comment on the relationship between `lcavol` and `lpsa`.

```
par(mfrow = c(1, 2)) #graphical parameters: two graphs per window
#Function plot is plot(x = , y = ) or plot(y ~ x)
#this works for vectors! (error message otherwise)
plot(exp(lpsa) ~ exp(lcavol),
xlab = "Cancer volume (milliliters per cc)", #y-axis label
ylab = "prostate specific antigen (ng/ml)", #x-axis label
main = "Prostate cancer dataset", #title
bty = "l", pch = 20) #bty: remove box, only x-y axis
#pch: type of plotting symbol (small filled circle)
plot(x = lcavol, y = lpsa,
xlab = "cancer volume (milliliters per cc), log scale",
ylab = "prostate specific antigen (ng/ml), log scale",
main = "Prostate cancer dataset",
bty = "l", pch = 20)

hist(exp(lcavol), xlab = "cancer volume (milliliters per cc)", main = "Histogram")
rug(exp(lcavol))
hist(exp(lpsa), xlab = "prostate specific antigen (ng/ml)", main = "Histogram")
rug(exp(lpsa))
```

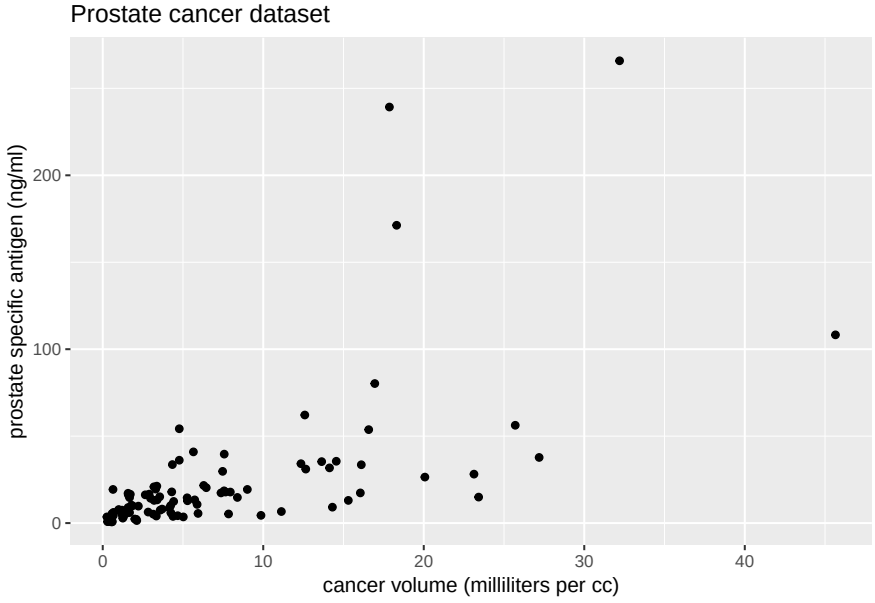
With ggplot2, the same graphs

```
library(ggplot2)

ggplot(data = prostate, aes(x = lcavol, y = lpsa)) +
```

```
  geom_point() +
  labs(y = "prostate specific antigen (ng/ml), log scale",
       x = "cancer volume (milliliters per cc), log scale",
       title = "Prostate cancer dataset")
```

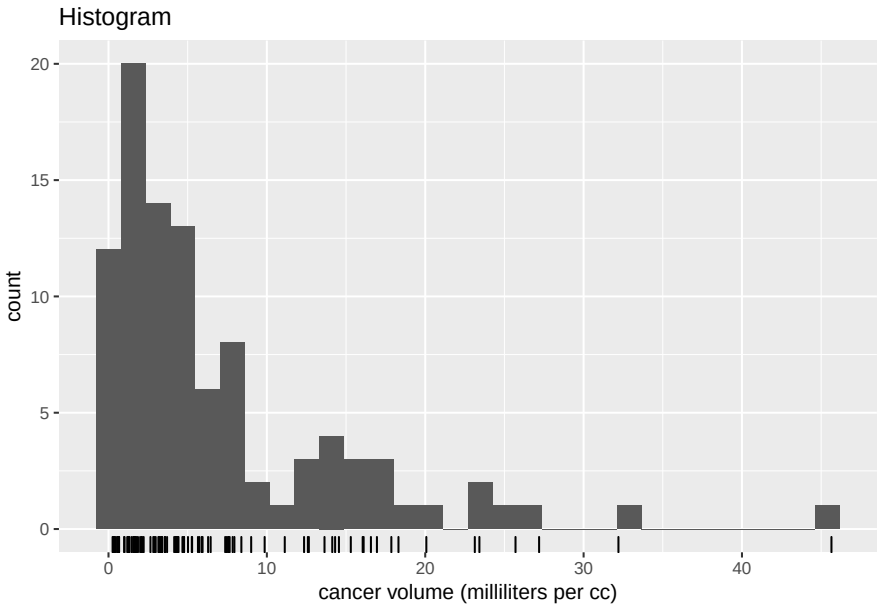



```
ggplot(data = prostate, aes(x = exp(lcavol), y = exp(lpsa))) +
  geom_point() +
  labs(y = "prostate specific antigen (ng/ml)",
       x = "cancer volume (milliliters per cc)",
       title = "Prostate cancer dataset")
```




```
ggplot(data = prostate, aes(x = exp(lcavol))) +
  geom_histogram(bins = 30) + geom_rug() +
```

```
  labs(x = "cancer volume (milliliters per cc)",
       title = "Histogram")
```



We can see that both variables are positive and positively skewed, so a log transform may lead to a more linear relationship, as indicated by the pairs plot. A multiplicative model on the original scale is thus reasonable.

d. Fit a linear model using the log prostate specific antigen as response variable, including a constant and the log cancer volume as covariates. Obtain numerically the OLS estimates $\hat{\boldsymbol{\beta}}$ of the parameters, the fitted values $\hat{\boldsymbol{y}}$ and the residuals $\boldsymbol{e}$ using the formulae given in class.

```
fit <- lm(lpsa ~ lcavol, data = prostate)
summary(fit)
```

```
##
## Call:
## lm(formula = lpsa ~ lcavol, data = prostate)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.67624 -0.41648  0.09859  0.50709  1.89672
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  1.50730    0.12194   12.36   <2e-16
## lcavol       0.71932    0.06819   10.55   <2e-16
##
## Residual standard error: 0.7875 on 95 degrees of freedom
## Multiple R-squared:  0.5394, Adjusted R-squared:  0.5346
## F-statistic: 111.3 on 1 and 95 DF,  p-value: < 2.2e-16
```

```
#Create response vector and design matrix
y <- lpsa
X <- cbind(1, lcavol)
```

```
#Create function to compute coefs "by hand"
coefs_vals <- function(x, y){
  c(solve(crossprod(x), crossprod(x, y)))
}
# Compute coefficients, fitted values and residuals
beta_hat <- coefs_vals(x = X, y = lpsa)
yhat <- c(X %*% beta_hat)
e <- y - yhat
```

The function `lm` fits a linear model by least squares to a dataset. The function `summary` will return coefficient estimates, standard errors and various other statistics and print them in the console.

The formula for `lm` must be of the form `y ~`, and any combination of the variables appearing on the right hand side of the `~` will be added as new columns of the design matrix. By default, the latter includes a column of ones. To remove it, use `+0` or `-1`. If you have two covariates `x1` and `x2`, the model `x1+x2` will have for ith row $(1, x_{i1}, x_{i2})$, while the model `x1+x2+x1:x2`≡`x1*x2` will include an *interaction* term `x1:x2`. The latter just means product, so the ith row of the design matrix would be $(1, x_{i1}, x_{i2}, x_{i1}x_{i2})$. **R** will drop any collinear vectors, warn you and report `NA` in the summary output.

e. Compare the quantities you obtained in the last question with the output of the function `lm`.

```
## [1] 1.5072975 0.7193204
```

```
## (Intercept)      lcavol
##   1.5072975   0.7193204
```

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

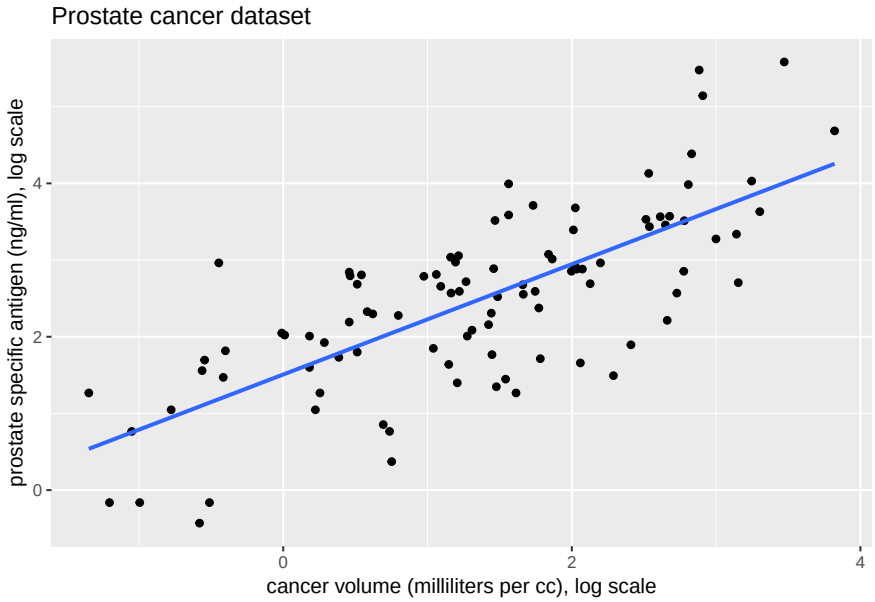
```
## [1] TRUE
```

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

f. Add the fitted regression line to the scatterplot of lcavol against lpsa .

```
par(mfrow = c(1, 1))
plot(lpsa ~ lcavol, data = prostate,
  xlab = "Cancer volume (milliliters per cc), log scale",
  ylab = "prostate specific antigen (ng/ml), log scale",
  main = "Prostate cancer dataset",
  bty = "l", pch = 20)
abline(fit, lwd = 2) #simply add regression line, lwd is line width
```

```
ggplot(data = prostate, aes(x = lcavol, y = lpsa)) +
  geom_point() +
  labs(y = "prostate specific antigen (ng/ml), log scale",
       x = "cancer volume (milliliters per cc), log scale",
       title = "Prostate cancer dataset") +
  geom_smooth(method = "lm", se = FALSE)
```



g. Interpret the changes in prostate specific antigen (not the log prostate specific antigen), including any units in your interpretations.

The interpretation is as follows. We fit

$$\log(\texttt{psa}_i) = \beta_0 + \beta_1 \log(\texttt{cavol}_i) + \varepsilon_i.$$

On the original scale, this translates into the multiplicative model $\texttt{psa}_i = \exp^{\beta_0} \texttt{cavol}_i^{\beta_1} \exp(\varepsilon_i)$. The effect of an increase of the volume of cancer of prostate cancer by one milliliter per cubic centimeter depends on the size of the latter of `cavol`, $(\texttt{cavol}_1/\texttt{cavol}_2)^{\beta_1}$ for levels $\texttt{cavol}_1$ and $\texttt{cavol}_2$. For example, an increase of the cancer volume from 2 ml per cc to 3 ml per cc leads to an increase of the concentration of PSA of 1.34 ng/ml.

h. Using the results of Exercise 4.2, obtain the orthogonal projection matrix $\mathbf{H}_\mathbf{X}$ and $\hat{\boldsymbol{\beta}}$ using a SVD decomposition (`svd`). Check your output.

```
#Hat matrix
Hmat <- X %*% solve(crossprod(X)) %*% t(X)
#SVD decomposition of X
svdX <- svd(X)
#OLS coefficients
beta_hat_svd <- svdX$v %*% (t(svdX$u) %*% lpsa / svdX$d)
Hmat_svd <- tcrossprod(svdX$u)
#Check that both quantities are equal
all.equal(Hmat, Hmat_svd, check.attributes = FALSE)
```

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

```
#use check.attributes = FALSE
#if you want to compare only the values
#and not e.g. the column names
all.equal(c(beta_hat_svd), beta_hat)
```

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

i. Compute the R_c^2 coefficient and compare with the one in summary output of the `lm` function. What can you say about the explanatory power of the covariate `lpsa`?

```
R2c <- sum((yhat-mean(y))^2)/sum((y-mean(y))^2)
R2c_lm <- summary(fit)$r.squared #this is centered version
all.equal(R2c, R2c_lm)
```

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

```
#Detach prostate from environment
detach(prostate)
```

The value of R_c^2 is about 0.54, so about half the variability can be explained by the model. There is reasonable explanatory power. Note that presence of cancer causes the prostate specific antigens to increase (not the other way around!). A linear model could nevertheless be sensible here if we wished to obtain a non-invasive detector for predicting presence/absence of cancer, assuming the antigen is present in blood samples, but that detection of cancer would require otherwise a biopsy.

j. Perform an explanatory data analysis of the `prostate` data set. Summarize your findings.

Here are some of the most important features we could detect by looking at the description, plots and summaries of the data set.

- Goal of the study: "PSA was proposed as a preoperative marker to predict the clinical stage of cancer";
- Individuals in the data set form a subset of the population of the study; the subset consists of men about to undergo radical prostatectomy. This implies they are in late stages of prostate cancer (see `gleason`);
- No missing values, no obvious outlier;
- Many variables are given on the log scale, potentially to remove the skewness (`lcavol`, `lweight`, `lcp`, `lbph`). This makes sense for volume (why?), less so for other variables;
- The most relevant explanatory variables are cancer volume (`lcavol`), weight (`lweight`) and SVI (`svi`);
- It is not clear why and how `pgg45` was constructed;
- 0.25 was added to benign prostatic hyperplasia and capsular penetration before taking the log-transform (to get `lbph` and `lcp`). It would perhaps be more adequate for interpretability to transform the capsular penetration back to the original scale;
- The weight of the tumor (`lweight`) is correlated with benign prostatic hyperplasia (consider an interaction term);
- Gleason is an ordered categorical, so it makes sense to cast it to a factor, with categories 6, 7 and (8,9). The seminal vesicle invasion (`svi`) is already binary;
- Obs. 37 is the only one with a Gleason score of 8 — keeping it leads to perfect fit for this data point and will lead to problems with cross-validation if `gleason` is included as a factor in the mean model;

Note that we cannot say anything about the distribution of the response `exp(lpsa)`, because the Gaussian linear model assumes the mean is $\mathbf{X}\boldsymbol{\beta}$ (so the skewness could be removed through the inclusion of covariates). Rather, one could fit both models on `exp(lpsa)` and `lpsa` scale and compare the diagnostics for the residuals.

Chapter 3

Frisch–Waugh–Lovell theorem

The FWL theorem has two components: it gives a formula for partitioned OLS estimates and shows that residuals from sequential regressions are identical.

Consider the following linear regression

$$\boldsymbol{y} = \mathbf{X}_1\boldsymbol{\beta}_1 + \mathbf{X}_2\boldsymbol{\beta}_2 + \boldsymbol{u},$$

where the response vector $\boldsymbol{y}$ is $n \times 1$, the vector of errors $\boldsymbol{u}$ is a realization from a mean zero random variable. The $n \times p$ full-rank design matrix $\mathbf{X}$ can be written as the partitioned matrix $(\mathbf{X}_1^\top, \mathbf{X}_2^\top)^\top$ with blocks $\mathbf{X}_1$, an $n \times p_1$ matrix, and $\mathbf{X}_2$, an $n \times p_2$ matrix. Let $\hat{\boldsymbol{\beta}}_1$ and $\hat{\boldsymbol{\beta}}_2$ be the ordinary least square (OLS) parameter estimates from running this regression. Define the orthogonal projection matrix $\mathbf{H}_\mathbf{X}$ as usual and $\mathbf{H}_{\mathbf{X}_i} = \mathbf{X}_i(\mathbf{X}_i^\top\mathbf{X}_i)^{-1}\mathbf{X}_i^\top$ for $i = 1,2$. Similarly, define the complementary projection matrices $\mathbf{M}_{\mathbf{X}_1} = \mathbf{I}_n - \mathbf{H}_{\mathbf{X}_1}$ and $\mathbf{M}_{\mathbf{X}_2} = \mathbf{I}_n - \mathbf{H}_{\mathbf{X}_2}$.

Theorem 3.1. *The ordinary least square estimates of $\boldsymbol{\beta}_2$ and the residuals from* (3) *are identical to those obtained by running the regression*

$$\mathbf{M}_{\mathbf{X}_1}\boldsymbol{y} = \mathbf{M}_{\mathbf{X}_1}\mathbf{X}_2\boldsymbol{\beta}_2 + \textit{residuals}.$$

In general, premultiplying both sides of the regression model by a projection matrix alters the model, so you will get different fitted values and residuals. Similarly, the model

$$\boldsymbol{Y} = \mathbf{X}_1\boldsymbol{\beta}_1 + \mathbf{X}_2\boldsymbol{\beta}_2 + \boldsymbol{\varepsilon}$$

is **not** equivalent to

$$\mathbf{M}_{\mathbf{X}_1}\boldsymbol{y} = \mathbf{M}_{\mathbf{X}_1}\mathbf{X}_2\boldsymbol{\beta}_2 + \mathbf{M}_{\mathbf{X}_1}\boldsymbol{\varepsilon}$$

because $\boldsymbol{\varepsilon}$ is not in $\mathbf{M}_\mathbf{X}$. This is true for the orthogonal decomposition

$$\mathbf{M}_{\mathbf{X}_1}\boldsymbol{y} = \mathbf{M}_{\mathbf{X}_1}\mathbf{X}_2\hat{\boldsymbol{\beta}}_2 + \mathbf{M}_{\mathbf{X}_1}\boldsymbol{e}$$

Below is an algebraic proof of the equality of the OLS coefficients. The following material is **optional**.

Proof. The easiest proof uses projection matrices, but we demonstrate the result for OLS coefficients directly. Consider an invertible $d \times d$ matrix $\mathbf{C}$ and denote its inverse by $\mathbf{D}$; then

$$\begin{pmatrix} \mathbf{C}_{11} & \mathbf{C}_{12} \\ \mathbf{C}_{21} & \mathbf{C}_{22} \end{pmatrix} \begin{pmatrix} \mathbf{D}_{11} & \mathbf{D}_{12} \\ \mathbf{D}_{21} & \mathbf{D}_{22} \end{pmatrix} = \mathbf{I}_p$$

gives the relationships

$$\begin{aligned}
\mathbf{C}_{11}\mathbf{D}_{11} + \mathbf{C}_{12}\mathbf{D}_{21} &= \mathbf{I}_{p_1} \\
\mathbf{C}_{11}\mathbf{D}_{12} + \mathbf{C}_{12}\mathbf{D}_{22} &= \mathbf{O}_{p_1,p_2} \\
\mathbf{C}_{22}\mathbf{D}_{21} + \mathbf{C}_{21}\mathbf{D}_{11} &= \mathbf{O}_{p_2,p_1} \\
\mathbf{C}_{22}\mathbf{D}_{22} + \mathbf{C}_{21}\mathbf{D}_{12} &= \mathbf{I}_{p_2}
\end{aligned}$$

from which we deduce that the so-called Schur complement of $\mathbf{C}_{22}$ is

$$\mathbf{C}_{11} + \mathbf{C}_{12}\mathbf{C}_{22}^{-1}\mathbf{C}_{21} = \mathbf{D}_{11}^{-1}$$

and

$$-\mathbf{C}_{22}\mathbf{C}_{21}(\mathbf{C}_{11} + \mathbf{C}_{12}\mathbf{C}_{22}^{-1}\mathbf{C}_{21})^{-1} = \mathbf{D}_{21}.$$

Substituting

$$\begin{pmatrix} \mathbf{C}_{11} & \mathbf{C}_{12} \\ \mathbf{C}_{21} & \mathbf{C}_{22} \end{pmatrix} \equiv \begin{pmatrix} \mathbf{X}_1^\top\mathbf{X}_1 & \mathbf{X}_1^\top\mathbf{X}_2 \\ \mathbf{X}_2^\top\mathbf{X}_1 & \mathbf{X}_2^\top\mathbf{X}_2 \end{pmatrix}$$

and plug-in this result back in the equation for the least squares yields

$$\begin{aligned}
\hat{\boldsymbol{\beta}}_1 &= (\mathbf{D}_{11}\mathbf{X}_1^\top + \mathbf{D}_{12}\mathbf{X}_2^\top)\boldsymbol{y} \\
&= \mathbf{D}_{11}(\mathbf{X}_1^\top - \mathbf{C}_{12}\mathbf{C}_{22}^{-1}\mathbf{X}_2)\boldsymbol{y} \\
&= \left(\mathbf{C}_{11} + \mathbf{C}_{12}\mathbf{C}_{22}^{-1}\mathbf{C}_{21}\right)^{-1}\mathbf{X}_1^\top\mathbf{M}_{\mathbf{X}_2}\boldsymbol{y} \\
&= (\mathbf{X}_1^\top\mathbf{M}_{\mathbf{X}_2}\mathbf{X}_1)^{-1}\mathbf{X}_1^\top\mathbf{M}_{\mathbf{X}_2}\boldsymbol{y}.
\end{aligned}$$

The proof that the residuals are the same is left as an exercise. □

The Frisch–Waugh–Lovell theorem dates back to the work of Frisch, R. and F. Waugh (1933)[1] and of M. Lovell (1963)[2].

3.1 Revisiting the interpretation of the parameters of a linear model

Geometrically, the linear model $\boldsymbol{y} = \mathbf{X}\boldsymbol{\beta}$ + residuals corresponds to the projection on to the span of $\mathbf{X}$ and gives the line of best fit in that space.

It is perhaps easiest to visualize the two-dimensional case, when $\mathbf{X} = (\mathbf{1}_n^\top, \mathbf{x}_1^\top)^\top$ is a $n \times 2$ design matrix and $\mathbf{x}_1$ is a continuous covariate. In this case, the coefficient vector $\boldsymbol{\beta} = (\beta_0, \beta_1)^\top$ represent, respectively, the intercept and the slope.

If $\mathbf{X} = \mathbf{1}_n$, the model only consists of an intercept, which is interpreted as the mean level. Indeed, the projection matrix corresponding to $\mathbf{1}_n$, $\mathbf{H}_{\mathbf{1}_n}$, is a matrix whose entries are all identically n^{-1}. The fitted values of this model thus correspond to the mean of $\boldsymbol{y}$, $\bar{y}$ and the residuals are the centred values $\boldsymbol{y} - \mathbf{1}_n\bar{y}$ whose mean is zero.

More generally, for $\mathbf{X}$ an $n \times p$ design matrix, the interpretation is as follows: a unit increase in x_{ij} ($\mathrm{x}_{ij} \mapsto \mathrm{x}_{ij} + 1$) leads to a change of β_j unit for y_i ($y_i \mapsto \beta_j + y_i$), other things being held constant. Beware of models with higher order polynomials and interactions: if for example one is interested in the coefficient for $\mathbf{x}_j$, but $\mathbf{x}_j^2$ is also a column of the design matrix, then a change of one unit in $\mathbf{x}_j$ will not lead to a change of $\beta_j x_j$ for y_j!

The FWL theorem says the coefficient $\boldsymbol{\beta}_2$ in the regression

$$\boldsymbol{y} = \mathbf{X}_1\boldsymbol{\beta}_1 + \mathbf{X}_2\boldsymbol{\beta}_2 + \boldsymbol{\varepsilon}$$

[1] https://www.jstor.org/stable/1907330
[2] https://doi.org/10.1080/01621459.1963.10480682

is equivalent to that of the regression

$$\mathbf{M}_1 \boldsymbol{y} = \mathbf{M}_1 \mathbf{X}_2 \boldsymbol{\beta}_2 + \boldsymbol{\varepsilon}$$

This can be useful to distangle the effect of one variable.

The intercept coefficient does not correspond to the mean of $\boldsymbol{y}$ unless the other variables in the design matrix have been centered (meaning they have mean zero). Otherwise, the coefficient β_0 associated to the intercept is nothing but the level of y when all the other variables are set to zero. Adding new variables affects the estimates of the coefficient vector $\boldsymbol{\beta}$, unless the new variables are orthogonal to the existing lot.

3.2 Factors

Oftentimes, the regressors that are part of the design matrix will be categorical variables, which can be ordered or unordered. In the design matrix, these will appear as matrix of binary indicators. In **R**, dummy variables (vectors of class `factor`) are used to indicate categorical variables, which are often encoded as strings or (perhaps worse) integers. The function `read.table` has an argument, `stringsAsFactors`, that will automatically cast strings to factors. Oftentimes, dummy indicators are encoded as integer vectors in data frames. There is a risk that the vector be interpreted as a continuous numeric if the levels are integers (for example, advancement of the state of an illness that is encoded as 1, 2, 3, ...).

Useful functions to deal with factors include

- `as.factor`: casts column vectors to factors;
- `is.factor`: to check whether the vector is a factor;
- `class`: reports the encoding (`factor` for factor objects);
- `summary`: displays counts of the various levels of a factor in place of the usual summary statistics;
- `levels`: the names of the different levels of a factor; can be used to replace existing category names by more meaningful ones;

The function `lm` knows how to deal with `factor` objects and will automatically transform it to a matrix of binary indiactors. For identifiability constraints, one level of the factor must be dropped and the convention in **R** is that the categories whose `levels` is first in alphabetical order is used as baseline. See `?factor` for more details.

Suppose your data set contains binary factors with only two levels and such that these are mutually exclusive. You may wish to merge these if they refer to the same variable, for example ethnicity. The following brute-force solution shows how to merge the factor vectors into a single factor. This will not change the model you get (since the design matrix would span the same space), but can affect the result of ANOVA calls since you will drop all different levels at once rather than subgroup by subgroup.

```
#Create dummy factors with different names to illustrate
a <- rbinom(500, size = 1, prob = 0.2)
b <- rep(0, 500)
b[a == 0] <- rbinom(sum(a == 0), size = 1, prob = 0.1)
#This is the output you get if they are encoded using 0/1
#Usually, they are columns of a data frame
newfactor <- data.frame(a = factor(a, labels = c("Hispanic","Other")),
                        b = factor(b, labels = c("Black", "Not black")))
#Make them have different levels (important that the Other class be encoded as zero
newlev <- cbind(as.numeric(newfactor$a) - 1, as.numeric(newfactor$b) - 1) %*% c(1,2)
mergfactor <- factor(newlev, levels = c("1","2","0"), labels = c("Hispanic", "Black", "Other"))
```

Ordered factors do not have the same parametrization as unordered ones, so be careful when interpreting them.

3.3 Example: seasonal effects

Suppose we are interested in estimating the seasonal model for quarterly data. Since each observation is recorded on one trimester, it could be postulated that the time of the year affect the response. This effect can be built in the design and tested for.

Consider a design matrix whose columns include $(\mathbf{S}_1, \mathbf{S}_2, \mathbf{S}_3, \mathbf{S}_4)$. The entries of the seasonal dummies are 0/1 depending on the season; for example, $S_{i1} = 1$ if the observation is recorded in the spring and $S_{i1} = 0$ otherwise, so that, for time ordered response vectors starting in the spring, we can write $\mathbf{S}_1 = (1,0,0,0,1,0,0,\ldots)^\top$ and $\mathbf{S}_1 + \mathbf{S}_2 + \mathbf{S}_3 + \mathbf{S}_4 = \mathbf{1}_n$. Thus, a regression model cannot include a mean and all four seasonal variables, but since $\mathcal{S}(\mathbf{S}_1, \mathbf{S}_2, \mathbf{S}_3, \mathbf{S}_4) = \mathcal{S}(\mathbf{S}_1, \mathbf{S}_2, \mathbf{S}_3, \mathbf{1}_n) = \mathcal{S}(\mathbf{S}_2, \mathbf{S}_3, \mathbf{S}_4, \mathbf{1}_n)$, any set of four vectors can be used.

If we drop the constant vector $\mathbf{1}_n$ from the regression (in **R**, by writing `0` or `-1` on the right hand side of the `lm` formula), the coefficients α_i, $i = 1, \ldots, 4$ in regression

$$\boldsymbol{y} = \mathbf{S}_1\alpha_1 + \mathbf{S}_2\alpha_2 + \mathbf{S}_3\alpha_3 + \mathbf{S}_4\alpha_4 + \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon}$$

correspond to the intercept for season i. If we include instead the dummy $\mathbf{S}_1$ and fit

$$\boldsymbol{y} = \mathbf{1}_n\gamma_1 + \mathbf{S}_2\gamma_2 + \mathbf{S}_3\gamma_3 + \mathbf{S}_4\gamma_4 + \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon},$$

then the constant for season 1 is $\alpha_1 = \gamma_1$, and is $\alpha_j = \gamma_1 + \gamma_j$ for season $j \in \{2,3,4\}$. The reference level is therefore the first season γ_1 and the coefficients $\{\gamma_j\}$, $j \in \{2,3,4\}$ are contrasts (the difference between the baseline and the seasonal level).

As just illustrated above, the interpretation of the coefficients when we have factors differs. Introducing factors lead to different intercepts for the different groups, whereas interactions with the factors lead to different slopes. Since we cannot include all levels of the factor as well as an intercept, the interpretation of the coefficient for the constant is the intercept of the baseline, i.e., the ommitted factor.

For simplicity, consider the simple case with an intercept and a single factor. Why do the coefficients β_1 and β_1' associated to the first columns of models

$$\mathbf{X}\boldsymbol{\beta} = \begin{pmatrix} \mathbf{1}_{n_1} & \mathbf{0}_{n_1} \\ \mathbf{1}_{n_2} & \mathbf{1}_{n_2} \end{pmatrix} (\beta_1 \quad \beta_2)^\top, \qquad \mathbf{X}'\boldsymbol{\beta}' = \begin{pmatrix} \mathbf{1}_{n_1} & \mathbf{0}_{n_1} \\ \mathbf{0}_{n_2} & \mathbf{1}_{n_2} \end{pmatrix} (\beta_1' \quad \beta_2')^\top$$

have the same interpretation? For the matrix $\mathbf{X}$ consisting of orthogonal regressors, this is clear. For the matrix $\mathbf{X}'$, recall the FWL theorem: the regression coefficient β_1' is the same as that of the regression $\mathbf{M}_{\mathbf{X}_2}\boldsymbol{y} = \mathbf{M}_{\mathbf{X}_2}\mathbf{1}_n + \boldsymbol{u}$ for $\mathbf{X}_2 = (\mathbf{0}_{n_1}^\top, \mathbf{1}_{n_2}^\top)^\top$. The matrix $\mathbf{M}_{\mathbf{X}_2}$ is a block matrix, whose first $n_1 \times n_1$ block contains entries n_1^{-1} and the rest of the entries is zero. $\mathbf{M}_{\mathbf{X}_2}\boldsymbol{y}$ does not affect the last n_2 entries of $\boldsymbol{y}$, while $\mathbf{M}_{\mathbf{X}_2}\mathbf{1}_n = \mathbf{1}_n - \mathbf{X}_2$. This reasoning generalizes to more complex settings with a slope and other regressors.

The discussion is illustrated on a dataset consisting of quarterly measurements of the gas consumption in the United Kingdom, from 1960 to 1986.

```
data(UKgas)
quarter <- rep(1:4, length = length(UKgas)) #create vector 1, 2, 3, 4, 1, ...
is.factor(quarter)
```

```
## [1] FALSE
```

```
quarter <- as.factor(quarter) #cast the vector to a factor
is.factor(quarter)
```

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

```
class(quarter)
```

```
## [1] "factor"
```

```
#What is name of the dummies
head(quarter)
```

```
## [1] 1 2 3 4 1 2
## Levels: 1 2 3 4
```

```
levels(quarter)
```

```
## [1] "1" "2" "3" "4"
```

```
levels(quarter) <- c("Q1", "Q2", "Q3", "Q4") #change names
```

The first model is of the form

$$\boldsymbol{y} = \mathbf{Q}_1\alpha_1 + \mathbf{Q}_2\alpha_2 + \mathbf{Q}_3\alpha_3 + \mathbf{Q}_4\alpha_4 + \boldsymbol{\varepsilon}.$$

```
gas_lm1 <- lm(log(UKgas) ~ quarter - 1)
coef(gas_lm1)
```

```
## quarterQ1 quarterQ2 quarterQ3 quarterQ4
##  5.989307  5.586806  5.039595  5.700242
```

```
head(model.matrix(gas_lm1), 10)
```

```
##    quarterQ1 quarterQ2 quarterQ3 quarterQ4
## 1          1         0         0         0
## 2          0         1         0         0
## 3          0         0         1         0
## 4          0         0         0         1
## 5          1         0         0         0
## 6          0         1         0         0
## 7          0         0         1         0
## 8          0         0         0         1
## 9          1         0         0         0
## 10         0         1         0         0
```

The model with all the quarterly dummies gives the quarterly average value $\exp(\alpha_j)$ in quarter j, $j = 1,\ldots,4$. If we include an intercept, the first factor is selected as baseline and the coefficients of the quarters `Q2` to `Q4` are contrasts. For this model, say

$$\boldsymbol{y} = \mathbf{1}_n\gamma_1 + \mathbf{Q}_2\gamma_2 + \mathbf{Q}_3\gamma_3 + \mathbf{Q}_4\gamma_4 + \boldsymbol{\varepsilon},$$

```
#New parameterization, with a constant
gas_lm2 <- lm(log(UKgas) ~ quarter) #R drops collinear by default and fits with Q2-Q4
coef(gas_lm2)
```

```
## (Intercept)   quarterQ2   quarterQ3   quarterQ4
##   5.9893074  -0.4025014  -0.9497127  -0.2890659
```

```
head(model.matrix(gas_lm2), 10)
```

```
##    (Intercept) quarterQ2 quarterQ3 quarterQ4
## 1            1         0         0         0
## 2            1         1         0         0
## 3            1         0         1         0
## 4            1         0         0         1
## 5            1         0         0         0
## 6            1         1         0         0
## 7            1         0         1         0
## 8            1         0         0         1
## 9            1         0         0         0
## 10           1         1         0         0
```

The estimated average gas consumption in millions of therms is $\exp(\widehat{\gamma}_1) = \exp(\widehat{\alpha}_1)$ for the first quarter, $\exp(\widehat{\alpha}_2) = \exp(\widehat{\gamma}_1 + \widehat{\gamma}_2)$ for the second quarter, etc.

We can check that the interpretation is correct.

```
isTRUE(all.equal((coef(gas_lm2)[1] + coef(gas_lm2)[-1]),
                 coef(gas_lm1)[-1], check.attributes = FALSE))
```

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

3.4 Solutions

3.4.1 Exercise 4.4

Consider the linear model

$$\boldsymbol{y} = \mathbf{X}_1\boldsymbol{\beta}_1 + \mathbf{X}_2\boldsymbol{\beta}_1 + \boldsymbol{\varepsilon}$$

and suppose that $\mathbf{X}_1$ includes a column of ones.

```
data(Auto, package = "ISLR")
y <- Auto$mpg
X1 <- cbind(1, Auto$horsepower)
X2 <- Auto$weight
```

a. Form the projection matrices $\mathbf{H}_{\mathbf{X}}$, $\mathbf{H}_1$, $\mathbf{H}_2$ and the complementary projection matrices (the functions `cbind`, `%*%`, `solve` and `t` may be useful).

```
data(Auto, package = "ISLR")
y <- Auto$mpg
X1 <- cbind(1, Auto$horsepower)
X2 <- Auto$weight
##Warning: output not displayed
#Projection matrices
X <- cbind(X1, X2)
#Helper functions
#Create a function: function(args){ ...}
#last line will be object returned (if not assigned)
```

```
#otherwise use `return( )`: see below for example
proj_mat <- function(x){
  x %*% solve(t(x) %*% x) %*% t(x)
}
coefs_vals <- function(x, y){
  coefs <- c(solve(t(x) %*% x) %*% t(x) %*% y)
  return(coefs)
  #`c` transform output from n x 1 matrix to n-vector
}
resid_vals <- function(y, pmat){
  c((diag(1, length(y)) - pmat)  %*% y)
  #diag(1, ...) creates identity matrix
}
fitted_vals <- function(y, pmat){
  c(pmat %*% y)
}

#Create projection matrices
H  <- proj_mat(X)
H1 <- proj_mat(X1)
H2 <- proj_mat(X2)
M  <- diag(nrow(X)) - H
M1 <- diag(nrow(X)) - H1
M2 <- diag(nrow(X)) - H2
```

b. Obtain the OLS estimates $\widehat{\boldsymbol{\beta}}_1, \widehat{\boldsymbol{\beta}}_2$
c. Use the projection matrices to obtain the fitted values $\widehat{\boldsymbol{y}}$ and the estimated residuals $\widehat{\boldsymbol{\varepsilon}}$.

```
#OLS coefficients
beta_hat <- coefs_vals(X, y)
beta_hat
#Fitted values
fitted_vals(y, H)
#Residuals
resid_vals(y, H)
```

d. What happens to the residuals if your regressors do not include a vector of constants?

If a constant is not included, the residuals are not centered unless the columns of the design matrix and the response were centered, meaning they had expectation zero. This is why a column vector of ones is always included unless the mean is known (from theory or otherwise) to be zero.

```
#Removing the row of constants
res_uncentered <- resid_vals(y, proj_mat(X[,-1])) #subset [row, column]
#X[,-1] take all rows, all columns but first
mean(res_uncentered)
```

```
## [1] 3.32711
```

```
#Consequence of not centering is that residuals are not mean zero
```

e. Verify numerically Frisch–Waugh–Lovell's theorem and test the different regression models from Exercice 2.4 to validate your answers.

```r
#The null models regression has
coef_0 <- coefs_vals(x = X, y = y)[ncol(X)]
res_0  <- resid_vals (y = y, pmat = H)
#The following function checks equality of the coefficients
#beta 2 is the last coef (here 1d)
check_FWL <- function(xmat, yvec, coef0 = coef_0, res0 = res_0){
  c(isTRUE(all.equal(coefs_vals(x = xmat, y = yvec)[ncol(xmat)], coef0)),
    isTRUE(all.equal(resid_vals(y = yvec, pmat = proj_mat(xmat)), res0))
  )
}

#Check the results of 4.3
res_mat <- cbind(check_FWL(yvec = y, xmat = X2), #1
check_FWL(yvec = H1 %*% y, xmat = X2), #2
check_FWL(yvec = H1 %*% y, xmat = H1 %*% X2), #3
check_FWL(yvec = H %*% y, xmat = X), #4
check_FWL(yvec = H %*% y, xmat = X2), #5
check_FWL(yvec = M1 %*% y, xmat = X2), #6
check_FWL(yvec = M1 %*% y, xmat = M1 %*% X2), #7 and 9
check_FWL(yvec = M1 %*% y, xmat = cbind(X1, M1 %*% X2)), #8
check_FWL(yvec = H %*% y, xmat = M1 %*% X2)) #10
colnames(res_mat) <-  c("(1)", "(2)", "(3)", "(4)", "(5)", "(6)", "(7,9)","(8)","(10)")
rownames(res_mat) <- c("coefficients","residuals")
print(res_mat)
```

```
##                (1)   (2)   (3)   (4)   (5)   (6) (7,9)  (8)  (10)
## coefficients FALSE FALSE FALSE  TRUE FALSE FALSE  TRUE TRUE  TRUE
## residuals    FALSE FALSE FALSE FALSE FALSE FALSE  TRUE TRUE FALSE
```

Chapter 4

Gaussian linear model

This section covers confidence and prediction intervals, diagnostic plots and quantile-quantile plots.

We present a worked-out example of a linear model fit to the `mtcars` data set

> The data was extracted from the 1974 Motor Trend US magazine, and comprises fuel consumption and 10 aspects of automobile design and performance for 32 automobiles (1973–74 models).

Residual plots are useful to diagnostic

- Misspecification of the response surface (nonlinearity, omitted variables)
- heteroscedasticity
- outliers
- autocorrelation (lack of independence of error terms) if observations are time ordered
- normality assumption.

4.1 Confidence and prediction intervals

In the linear model with IID errors $\boldsymbol{\varepsilon} \sim \mathsf{IID}(0, \sigma^2)$, we have $\mathsf{Var}(\hat{\boldsymbol{\beta}}) = \sigma^2(\mathbf{X}^\top\mathbf{X})^{-1}$. The standard errors for $\hat{\boldsymbol{\beta}}$ are then simply the square root of the diagonal entries (which are the variance $\mathsf{Var}(\hat{\beta}_j)$ for $j = 1, \ldots, p$. Confidence intervals for the coefficients are given by $\hat{\beta}_i \pm t_{n-p}(0.025)\mathsf{se}(\hat{\beta}_i)$.

We can also draw intervals around the regression line by considering combinations $\mathbf{x} = (1, \texttt{mpg})$ for different values of `mpg` as illustrated below. The reasoning is similar, except that we now obtain the interval for a function of $\widehat{\boldsymbol{\beta}}$. For each new vector of regressors $\mathbf{x}^i \equiv \mathbf{c}$, we get new fitted values $\hat{y}^i = \mathbf{x}^i\widehat{\boldsymbol{\beta}}$ whose variance is, by the delta-method, given by $\mathsf{Var}(y^i) = \sigma^2\mathbf{x}^{i\top}(\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{x}^i$. We replace σ^2 by the usual estimator s^2 and thus the pointwise confidence interval is given by the usual Student-t test statistics, with this time

$$\hat{y}^i \pm t_{n-p}(0.025)\mathsf{se}(\hat{y}^i) = \mathbf{x}^i\widehat{\boldsymbol{\beta}} \pm t_{n-p}(0.025)\sqrt{s^2\mathbf{x}^{i\top}(\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{x}^i}.$$

For the prediction interval, we consider instead

$$\mathbf{x}^i\widehat{\boldsymbol{\beta}} \pm t_{n-p}(0.025)\sqrt{s^2\left[\mathbf{I}_n + \mathbf{x}^{i\top}(\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{x}^i\right]}.$$

Provided the model is correct, new observations y_{new} should fall 19 times out of 20 within the reported prediction interval.

As we move away from the bulk of the data (average value of $\mathbf{x}$), the hyperbolic shape of the intervals becomes visible. Note here how the prediction interval is necessarily wider than the confidence interval (iterated variance formula).

```r
# The function lm and its output
ols <- lm(mpg ~ wt, data = mtcars)
res <- resid(ols)
X <- cbind(1,  mtcars$wt)
n <- nrow(X)
s2 <- sum(res^2) / (length(res) - ncol(X))
std_err <- sqrt(diag(s2 * solve(crossprod(X))))
beta_hat <- coef(ols)
#Covariance matrix of (beta0, beta1)
Sigma <- vcov(summary(ols))
#check the calculation
isTRUE(all.equal(Sigma, s2 * solve(crossprod(X)), check.attributes = FALSE))
```

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

```r
#Standard error of estimates
std_err <- sqrt(diag(Sigma))
#Extract leverage values h_ii
leverage <- as.vector(hatvalues(ols))
# Compare with manual calculation from diagonal matrix
leverage_man <- rep(0, n)
XtXinv <- solve(crossprod(X))
for(i in 1:n){
  leverage_man[i] <- X[i,] %*% XtXinv %*% X[i,]
}
isTRUE(all.equal(leverage, leverage_man))
```

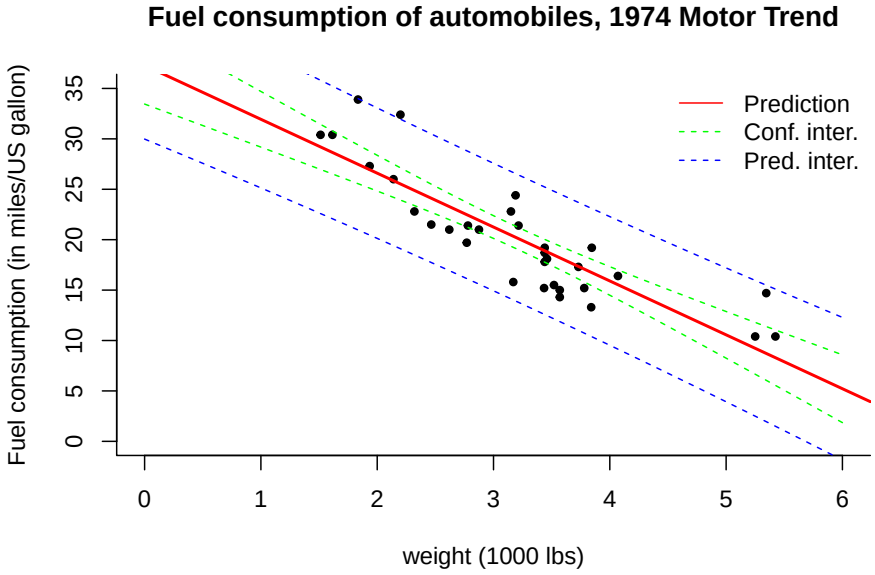
```
## [1] TRUE
```

```r
# Plot data set
plot(mpg ~ wt, data = mtcars,
     xlab = "weight (1000 lbs)",
     ylab = "Fuel consumption (in miles/US gallon)",
     main = "Fuel consumption of automobiles, 1974 Motor Trend",
     bty = "l", pch = 20, ylim = c(0, 35), xlim = c(0, 6))
abline(beta_hat, col = 'red', lwd = 2)

#Confidence intervals
tqu = qt(0.975, df = nrow(X)- ncol(X))
conf_inter <- cbind(beta_hat - tqu * std_err, beta_hat + tqu * std_err)
#Compare with lm output
confint(ols)
```

```
##                 2.5 %    97.5 %
## (Intercept) 33.450500 41.119753
## wt          -6.486308 -4.202635
```

```r
xstar <- seq(0, 6, by = 0.1)

#Confidence interval for prediction using lm output
ystar_confint <- predict(ols, newdata = data.frame(wt = xstar), interval = 'confidence')
lines(xstar, ystar_confint[ ,2], lty = 2, col = 'green')
```

```
lines(xstar, ystar_confint[ ,3], lty = 2, col = 'green')
#Prediction interval using lm output
ystar_predint <- predict(ols, newdata = data.frame(wt = xstar), interval = 'prediction')
lines(xstar, ystar_predint[ ,2], lty = 2, col = 'blue')
lines(xstar, ystar_predint[ ,3], lty = 2, col = 'blue')
legend(x = "topright", col = c("red", "green","blue"),
       lty = c(1, 2, 2), bty = "n",
       legend = c("Prediction","Conf. inter.","Pred. inter."))
```



The function `predict` takes as imput a `data.frame` object containing the same column names as those of the fitted `lm` object. The names can be obtained from `names(ols$model)[-1]`.

As usual, we can verify we get the same result if we computed the intervals manually.

```
#Manually (see class notes)
confint_xstar <- tqu * sqrt(s2 * apply(cbind(1, xstar), 1, function(cvec){t(cvec) %*% solve(crossprod(X
fitted_xstar <- cbind(1, xstar) %*% cbind(beta_hat)
lines(xstar,  fitted_xstar - confint_xstar, lty = 2, col = 'green')
lines(xstar,  fitted_xstar + confint_xstar, lty = 2, col = 'green')
```

4.2 Residuals

There are many types of residuals. The model residuals are simply $\boldsymbol{e} = \mathbf{M}_{\mathbf{X}}\boldsymbol{y}$, which can be obtained through `resid` for `lm` objects. We can verify numerically that $\hat{\boldsymbol{y}} \perp \boldsymbol{e}$ and verify that $\mathbf{X}^\top \boldsymbol{e} = \mathbf{0}_p$.

```
#Fitted values
yhat <- fitted(ols)
#Residuals
e <- resid(ols)

#Orthogonality (by construction)
isTRUE(all.equal(c(e %*% yhat), 0))
```

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

```
isTRUE(all.equal(c(e %*% X), rep(0, 2)))
```

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

In the sequel, we will look at calculation of various variants of the residuals. The first are the standardized residuals, also internally studentized residuals. These are defined as $r_i = e_i/\{s(1-h_{ii})^{1/2}\}$, i.e. each residual e_i is scaled by its individual variance to create homoscedastic residuals r_i.

```
#we divide so they have the same variance - but not independent
r <- e/sqrt(s2*(1-leverage))
#also obtainable via rstandard(ols)
isTRUE(all.equal(rstandard(ols), r))
```

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

Because the ith residual is used in both the numerator and in the denominator (in the calculation of s^2), the standardized (internally studentized) residual follows marginally an approximate scaled Student distribution. However, because of the use of s^2 in the denominator, the entries of $\boldsymbol{r}$ are bounded by $\pm n-p$. They are also not independent, even if this fact is often omitted in practice. While they will be approximately centered (with mean zero and variance one), they can (and should) be recentered before undertaking visual diagnostics.

The externally studentized residuals t_i are obtained by excluding the ith observation from the calculation of the variance. The advantage of doing this is that $\{t_i\}_{i=1}^n$ are marginally Student distributed with $n-p-1$ degrees of freedom (but they are again not independent). These are typically the residuals that are displayed in Q-Q plots. The externally studentized residuals can be obtained with the function `rstudent`.

We will derive formulas for $\hat{\boldsymbol{\beta}}_{-i}$, s^2_{-i}, Cook distance and t_i later in the exercises. Two of these are used below, namely

$$t_i = \frac{e_i}{[s^2_{-i}(1-h_{ii})]^{1/2}}, \qquad s^2_{-i} = \frac{(n-p)s^2 - e_i^2/(1-h_{ii})}{n-p-1}.$$

```
#Externally studentized residuals
smi <- influence(ols)$sigma
s2mi <-((n-2)*s2-e^2/(1-leverage))/(n-3)
isTRUE(all.equal(smi^2, s2mi))
```

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

```
esr <- e/sqrt(s2mi*(1-leverage))
isTRUE(all.equal(rstudent(ols), esr))
```

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

The last type of residual is the leave-one-out cross validation residual. These are the residuals obtained by fitting the linear model to all observations, but the ith, i.e., $\boldsymbol{y}_{-i} = \mathbf{X}_{-i}\boldsymbol{\beta} + \boldsymbol{\varepsilon}$. Let $\hat{\boldsymbol{\beta}}_{-i}$ denote the OLS coefficients from this regression and $\hat{y}_{i,-i} = \mathbf{x}_i\hat{\boldsymbol{\beta}}_{-i}$ the predicted value for the left-out $\mathbf{x}_i$ regressor. The ith leave-one-out cross validation residual is $e_{i,-i} = y_i - \hat{y}_{i,-i} = e_i/(1-h_{ii})$. We can use these to calculate the PRESS statistic, $\text{PRESS} = \sum_{i=1}^n e^2_{i,-i}$

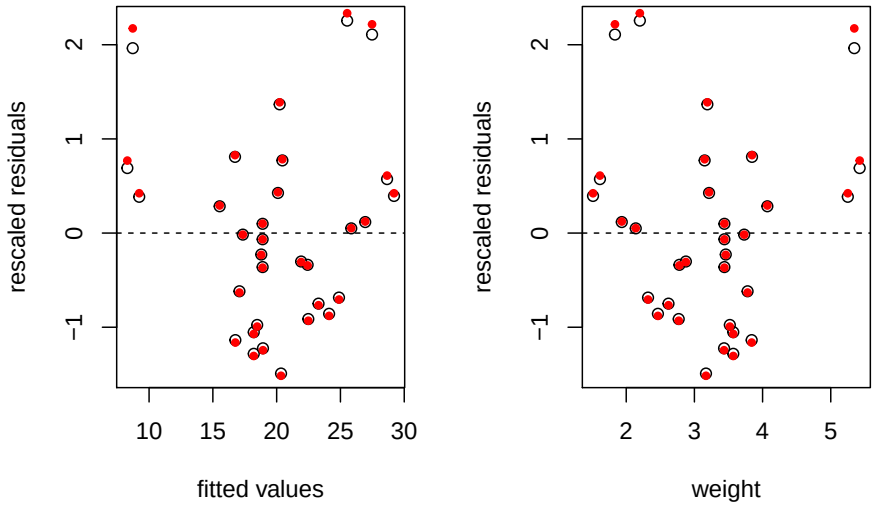
```
# LOOCV residuals e/(1-leverage)
loocv <- e/(1-leverage)
loocv_err <- rstandard(ols, type = "pred")
PRESS <- crossprod(loocv_err)
```

4.3 Diagnostic plots

If the underlying model is truly linear, a plot of $\boldsymbol{e}$ against $\hat{\boldsymbol{y}}$, should be flat because the two are by construction orthogonal. In practice, we rescale $\boldsymbol{e}$ by s to ensure that the variance is closer to unity. If there are omitted higher-order interactions, these will show up in such a plot.

In practice, there is often little difference between the rescaled residuals $\boldsymbol{e}/s$ and the internally studentized residuals $\boldsymbol{r}$. The former are orthogonal to $\hat{\boldsymbol{y}}$, while the latter have equal variance.

```
par(mfrow = c(1, 2)) #split the graphic window (1 row, 2 columns)
#Fitted values vs raw residuals/s2
plot(y = e/sqrt(s2), x = yhat,
     xlab = "fitted values", ylab = "rescaled residuals"); abline(h = 0, lty = 2)
#Fitted values vs internally studentized residuals
points(y = r, x = yhat, pch = 20, col = 2)
#Regressor weight vs residuals
plot(y = e/sqrt(s2), x = X[,2], xlab = "weight",
     ylab = "rescaled residuals"); abline(h = 0, lty = 2)
points(y = r, x = mtcars$wt, pch = 20, col = 2)
```




```
#graphics.off()
par(mfrow = c(1, 1))
```

An alternative is `residualPlot(lm(mpg ~ hp + wt, data = mtcars))`, which adds the line for a quadratic regression of $\hat{\boldsymbol{y}}$ against standardized residuals.

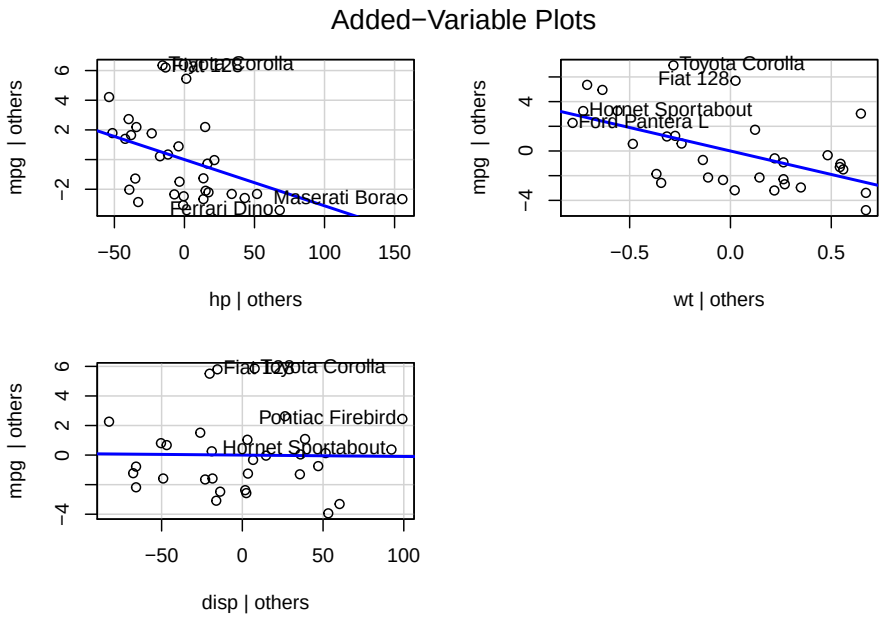
4.3.1 Added-variable plots

We can assess graphically whether a regressor should be included or not in the model. If the omitted regressor $\mathbf{X}_2$ is redundant, its coefficient should be zero and we can project onto the orthogonal complement of the remaining regressors $\mathbf{M}_{\mathbf{X}_1}$ and the response to get the regression FWL for $\boldsymbol{\beta}_2$. The relationship between the two should have zero slope. The package `car` has a function `avPlot`.

In the regression of fuel consumption as a function of weight, we have not included the potentially important regressor `hp`, which measures the power of the engine. The added variable plot shows that it is an important

explanatory variable. In contrast, the displacement `disp` is either uncorrelated with `mpg` or its effect is already explained by `wt` and `hp`.

```
#install.packages("car")
library(car)
car::avPlots(model = lm(mpg ~ hp + wt + disp, data = mtcars))
```



4.3.2 Diagnostic of heteroscedasticity

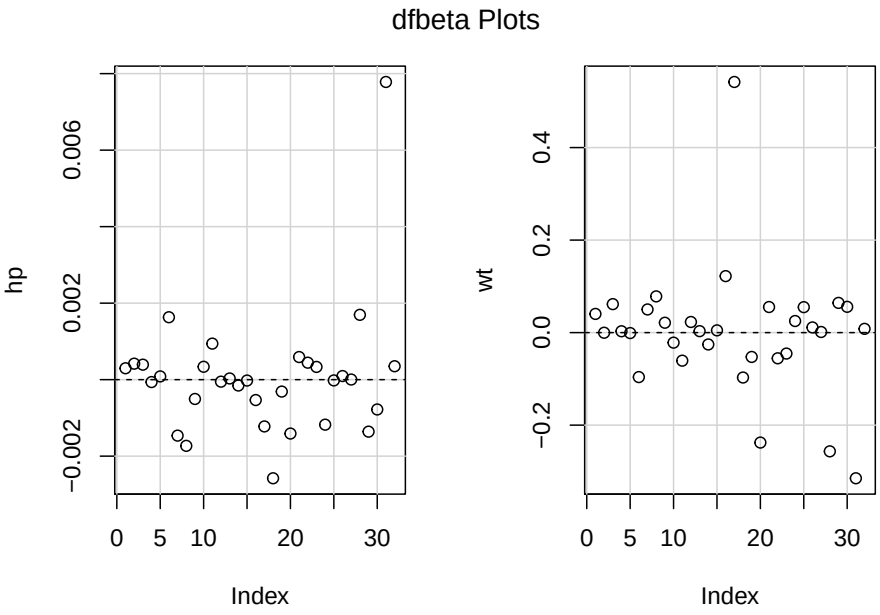
Unequal variance will often show up in time series. For example, many economic models postulate exponential growth, but this effect can appear linear at a small scale. However, the variance will not be constant and typically increase with the level of the observations. If there are factors, these may have different variances. A simple boxplot of the fitted values against the factor can flag heteroscedasticity.

4.3.3 Outliers

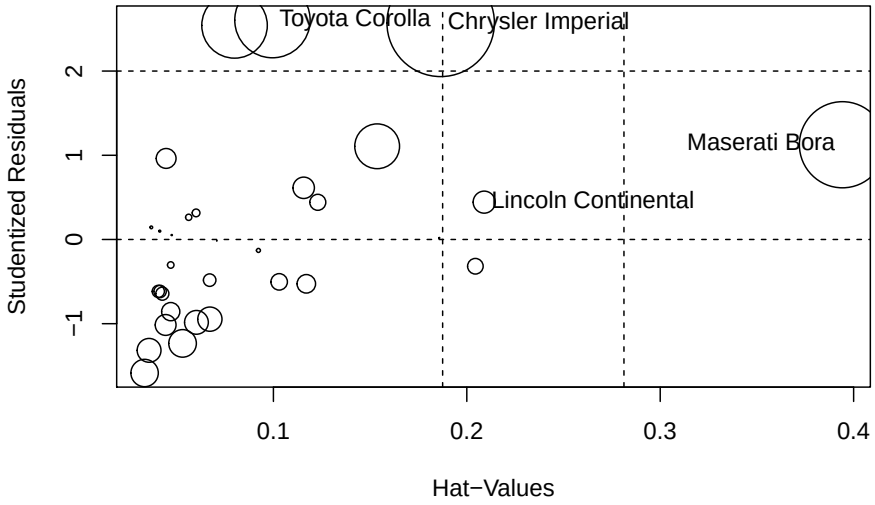
If an outlier is present and it has high leverage, it will draw the regression line towards itself. One way of assessing this (assuming there is a single such point) is to compute $\hat{\boldsymbol{\beta}}$ by fitting the model to all but the observation y_i. The difference between this estimate $\hat{\boldsymbol{\beta}}_{-i}$ and $\hat{\boldsymbol{\beta}}$ is called difference of betas, or `dfbeta`. We can compute the effect of the deletion efficiently (details later on this) and similarly rescale the estimates to get a standardized difference.

We can also look at the Cook's distance, the leverage values and the externally studentized residuals. These are often combined in a bubble pplot in which the radius of the circle is proportional to Cook's distance, with the leverage on the x-axis and the value of the externally studentized residuals $\boldsymbol{t}$ on the y-axis.

```
dfbetaPlots(model = lm(mpg ~ hp + wt, data = mtcars))
```




```
influencePlot(model = lm(mpg ~ hp + wt, data = mtcars))
```




```
##                       StudRes        Hat     CookD
## Lincoln Continental 0.4434296 0.20897838 0.0178091
## Chrysler Imperial   2.5724776 0.18648721 0.4236109
## Toyota Corolla      2.6051516 0.09950335 0.2083933
## Maserati Bora       1.1250084 0.39420816 0.2720397
```

4.4 Quantile-quantile plots

The distributional assumption is mostly assessed using quantile-quantile plots. However, the latter are hardly useful unless we superimpose some confidence intervals to the graph. We will cover two methods for producing Q-Q plots for linear models: one using an orthogonal transformation that makes the estimated residuals IID. The second uses the externally studentized residuals.

4.4.1 Quantile-quantile plot of externally studentized errors

Recall that the quantile-quantile plot has

- on the x-axis, the theoretical quantiles, $F^{-1}(\text{rank}(X_i)/(n+1))$
- on the y-axis, the empirical quantiles, X_i

For a Gaussian Q-Q plot, we will need to estimate both the mean and the variance. The usual estimators will do, replacing σ^2 with s^2 in the calculations, but all results will be approximate. One can obtain standard residuals by subtracting the mean and scaling by the standard deviation (using e.g. the function `scale`). The function `qqnorm` plots a Normal Q-Q plot without rescaling and the function `qqline` adds a line passing through the first and third quartile. Since these are robust estimates, this is a sensible option but implies that the scales of the Q-Q plot are not the same on the x-axis than on the y-axis. It is preferable to use these estimates to rescale the data, so as to facilitate the inclusion of approximate confidence intervals.

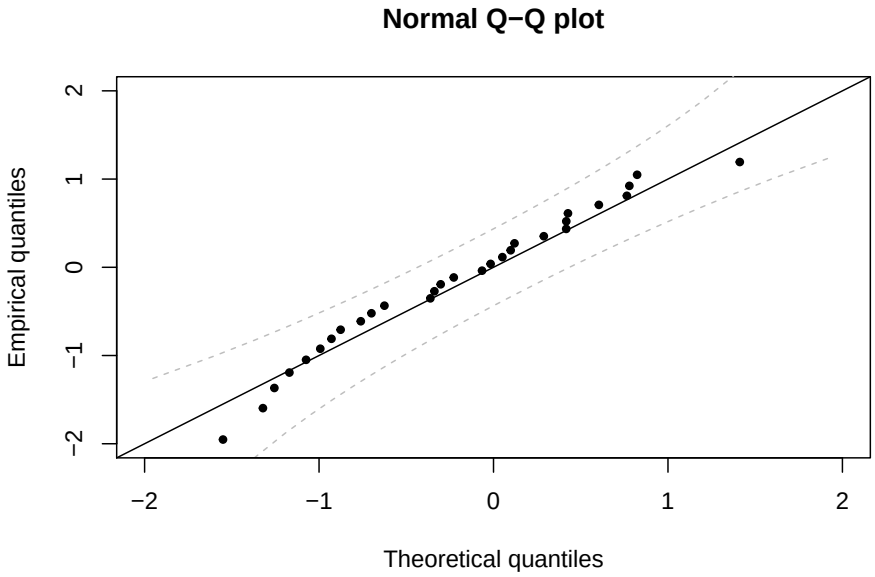
We now compute pointwise confidence intervals using the result on the distribution of the order statistic, which will be covered in Exercise 9.2 (in 2018).

Suppose $\{X_i\}_{i=1}^n$ are independent random variables with absolutely continuous distribution function F and density f. Let $X_{(k)}$ denote the kth order statistic: $X_{(1)} \leq \cdots \leq X_{(n)}$; then $F(X_{(k)})$ follows a Beta distribution with parameters k and $n+1-k$. Let $\mathfrak{b}_\eta$ denote the η-quantile of the $\text{Beta}(k, n+1-k)$ distribution. Then,

$$\Pr\left\{\mathfrak{b}_{\alpha/2} \leq F(X_{(k)}) \leq \mathfrak{b}_{1-\alpha/2}\right\} = 1-\alpha$$

so an approximate confidence interval for $X_{(k)}$ is $[F^{-1}(\mathfrak{b}_{\alpha/2}), F^{-1}(\mathfrak{b}_{1-\alpha/2})]$.

```
#Student plotting position F^(-1)(E[U_{(i)}])
emp_quant <- qt(rank(esr)/(n + 1),  df = n - 3)
#Function to compute the pointwise confidence intervals
#You can simply copy-paste this for your own plots
confint.qqplot.ptw <- function(n, dist = "norm", ...){
  t(sapply(1:n, function(i){
  #Beta order statistic quantiles, mapped to Student scale
    do.call(paste0('q', dist), list(qbeta(c(0.025, 0.975), i, n - i + 1), ...))
  }))
}

#Call the function
confint_lim <- confint.qqplot.ptw(n = n, dist = "t", df = n - 3)
#Plot these confidence bands alongside with the empirical quantile plotting position
matplot(sort(emp_quant), confint_lim, type = "l", lty = 2, col="grey",
        main = "Normal Q-Q plot", xlim = c(-2, 2), ylim = c(-2, 2),
        xlab = "Theoretical quantiles", ylab = "Empirical quantiles")
#Theoretical line of fit
abline(a = 0, b = 1)
#Add observations
points(esr, emp_quant, pch = 20)
```



4.4.2 Quantile-quantile plot using the QR decomposition

The problem with the residuals is that, while $\boldsymbol{e}$ are normally distributed with variance $\sigma^2\mathbf{M}_\mathbf{X}$, they are linearly dependent (think of the constraint $\mathbf{X}^\top\boldsymbol{e}=\mathbf{0}_p$).

Therefore, $\mathbf{M}_\mathbf{X}$ is not invertible (it is an $n\times n$ matrix of rank $n-p$) — `solve(diag(n) - Hmat)` typically returns an error message although some matrix decomposition such as the SVD handle the rank deficient case. One can use an orthogonal transformation to obtain a set of $n-p$ independent residuals, but it is then difficult to relate these to the regressors.

One such orthogonal transformation is provided by the QR decomposition, $\mathbf{X}=\mathbf{QR}$ where $\mathbf{Q}$ is an orthogonal matrix. Consider the linear model

$$\mathbf{Q}^\top\boldsymbol{Y}=\mathbf{Q}^\top\mathbf{X}\boldsymbol{\beta}+\boldsymbol{u};$$

the last $n-p$ estimated residuals of the vector $\tilde{\boldsymbol{e}}=\mathbf{Q}^\top\boldsymbol{e}$ will be IID Normal and the first p identically zero. In R, use the function `t(qr.Q(qr(X), complete = TRUE))` to obtain the matrix $\mathbf{Q}^\top$ associated to the design matrix `X`.

Note that it is difficult to detect violation of the normality assumption because observations that arise from distributions that are not heavy-tailed still behave roughly like they are normally distributed when we scale them. This phenomenon, *supernormality*, is a consequence of the central limit theorem.

4.4.3 Monte Carlo methods for confidence intervals

This section contains **optional** material. It contains advanced material that can be skipped upon first reading.

An alternative to asymptotic theory (which may be unreliable in small samples) is to rely on simulations. The idea is to obtain a statistic whose distribution is (at least asymptotically) pivotal, i.e. fully specified under the null hypothesis. One can simulate samples from the null distribution B times and compare the resulting data points with the test statistic calculated from the observed sample. This method, which is termed bootstrap test, is particularly powerful when we want to obtain critical values for test statistics, like e.g. $\max(|t_i|)$, whose distribution is untractable.

Under the null hypothesis of the Gaussian linear model, $\{y_i\}_{i=1}^n$ is a simple random sample from a Gaussian distribution $\boldsymbol{Y}\sim\mathcal{N}_n(\mathbf{X}\boldsymbol{\beta},\sigma^2\mathbf{I}_n)$. One can resort to simulations to obtain approximate confidence intervals at asymptotic level α. Specifically, the postulated data generating mechanism is

$$\boldsymbol{Y}=\mathbf{X}\boldsymbol{\beta}+\boldsymbol{\varepsilon}.$$

We will replace the unknown parameters (here $\boldsymbol{\beta}$ and σ^2) by their best linear unbiased estimate. For $b = 1, \ldots, B$ where $B/\alpha \in \mathbb{N}$, repeat the following steps:

1. sample $\boldsymbol{\varepsilon}_b \sim \mathcal{N}_n(\mathbf{0}_n, s^2\mathbf{I}_n)$ and form $\boldsymbol{y}_b = \mathbf{X}\hat{\boldsymbol{\beta}} + \boldsymbol{\varepsilon}_b$.
2. run least squares with the design matrix $\mathbf{X}$ and extract the residuals $\boldsymbol{e}_b$. Compute the centered externally Studentized version.
3. sort the samples and the $\alpha/2$ and $1-\alpha/2$ empirical quantiles of each vector of order statistics

This provides a pointwise confidence interval for each order statistic. We can assess the overall coverage of the intervals by checking whether or not one of the points falls outside the confidence envelope. Since we have B datasets, we can check each in turn (using the others as reference for the interval) and check the fraction that have at least one observations outside the simulated pointwise bands. This gives a measure of the overall error rate. We can adjust k until we get the correct overall empirical coverage.

The calculation is rather simple.

- calculate the rank of each observation (column by column) in the $B \times n$ matrix of simulated points.
- an exceedance occurs if and only if the rank of an observation in a line is below or equal to k, or at least $B+1-k$.
- to check this, it suffices to retain the minimum and maximum rank of each row.

These methods are implemented in the package `boot` and returned by the function `boot::envelope`; you must supply a matrix of replicates test statistics. In our setting, these are the ordered samples from the externally studentized residuals

$$\left\{\left\{t^b_{(i)}\right\}_{i=1}^n\right\}_{b=1}^B.$$

You should choose B so that $B+1$ is divisible by α and rather large. $B = 9999$ should work well and not be too computationally costly for small datasets.

```
#Dimensions of the design matrix
n <- nrow(model.matrix(ols))
p <- ncol(model.matrix(ols))
#Bootstrap setting
B <- 9999
X <- model.matrix(ols)
betahat <- coef(ols)
boot_samp <- matrix(NA, nrow = B, ncol = n)
for(b in 1:B){
  #Generate new errors
  eps_samp <- rnorm(n, sd = sqrt(s2))
  Xbeta <- X %*% betahat
  #Create new replicate dataset
  yb <- Xbeta + eps_samp
  #Obtain externally studentized residuals
  #Sort them in increasing order
  boot_samp[b, ] <- sort(rstudent(lm(yb ~ -1 + X)))
}

#Add the dataset to the replicates
res_samp <- rbind((esr <- rstudent(ols)), boot_samp)
#Compute the quantiles of this experiment => per column means for each order statistic
confint_pw <- t(apply(res_samp, 2, quantile, probs = c(0.025, 0.975)))
#Alternatively, could sort each column and pick the k and B-k-1 entries

#Computed automatically by package bootstrap
```

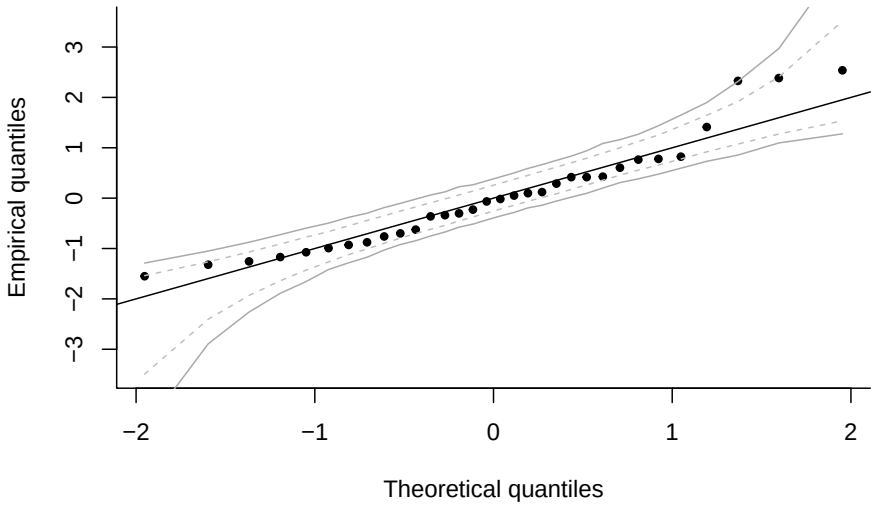
```
env <- boot::envelope(mat = boot_samp)

#Plot the confidence interval
matplot(y = cbind(sort(esr), confint_pw), x = qt((1:n) / (n + 1), df = n - p - 1),
       lty = c(1, 2, 2), col = c(1, "grey", "grey"), type = c("p","l","l"),
       pch = 20, xlab = "Theoretical quantiles", ylab = "Empirical quantiles", bty = 'l')
abline(a = 0, b = 1)

#Simultaneous confidence interval
#In how many of the replicates datasets do we have
#observations outside of the pointwise confidence bands?
R <- nrow(boot_samp)
alpha <- 0.05
k <- alpha * (R + 1)/2
#Simply check this as follows:
#For each column, return the rank of the simulation
rank_boot <- apply(boot_samp, 2, rank)
#For each row, keep minimum and maximum rank
minmax_rk <- t(apply(rank_boot, 1, function(x){c(min(x), max(x))}))
#Go outside of the pointwise confidence interval if
#min(rank) < k or max(rank) > R + 1 - k
emp_boot_cov <- function(k){
 1-mean((I(minmax_rk[,1] > k))*I(minmax_rk[,2] < (R+1-k)))
}
#In how many of the replicates datasets do
#we have observations outside of the bounds?
emp_boot_cov(k)
```

```
## [1] 0.5731573
```

```
#Ouch! decrease k until we hit alpha percentage of exceedances (0.05)
boot_cov_k <- sapply(1:k, emp_boot_cov)
klev <- match(TRUE, boot_cov_k > alpha) - 1
if(klev == 0){
 klev <- 1
}
env_jt <- apply(boot_samp, 2, sort)[c(R+1-klev, klev),]

#This is what is returned by function envelope
isTRUE(all.equal(env_jt, env$overall))
```

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

```
matplot(x = qt((1:n)/(n+1), df = n - p -1), y = t(env$overall),
       col = "darkgrey", add = TRUE, type = "l", lty = 1)
```



4.4.4 Parametric bootstrap confidence intervals using the QR decomposition

This section contains **optional** material.

There is an alternative way to obtain pointwise (and even simultaneous) confidence intervals for the QR decomposition. Under the null hypothesis: $\boldsymbol{\varepsilon} \sim \mathcal{N}_n(\mathbf{0}_n, \sigma^2\mathbf{I}_n)$, we get $\tilde{\boldsymbol{\varepsilon}} \sim \mathcal{N}_{n-p}(\mathbf{0}_n, \sigma^2\mathbf{I}_n)$ and therefore $(\tilde{\boldsymbol{\varepsilon}} - \bar{\tilde{\boldsymbol{\varepsilon}}})/\text{sd}(\tilde{\boldsymbol{\varepsilon}}) \dot{\sim} \mathcal{N}_{n-p}(\mathbf{0}_n, \mathbf{I}_n)$ is asymptotically pivotal. A pivotal quantity has a fully specified distribution.

We have only observed one sample, so comparisons are difficult because the measurements are limited. Under the null hypothesis, it is however easy to generate new datasets: simply generate new observations $\tilde{\boldsymbol{\epsilon}}_b \sim \mathcal{N}_{n-p}(\mathbf{0}_n, \mathbf{I}_n)$ and standardize them, mimicking what we have done to obtain our sample quantiles. This gives us a potentially unlimited number of samples to compare our observations to. By ordering each new set of errors of the B replicates, we get a matrix of observations whose rows are order statistics from a run and whose columns corresponds to the empirical distribution of each order statistic. A symmetric 95% confidence interval is obtained by taking the empirical (0.025, 0.975) percentiles of each order statistic.

4.5 Solutions

4.5.1 Exercise 7.1 - Study of growth hormones

We will use `factor` objects in the sequel. A `factor` encodes a matrix of binary variables, potentially identied using strings, so that the output is readable. **R** know how to handle the vector if it is passed to e.g. the function `lm`. By default, if the matrix spans $\mathbf{1}_n$, the first level (in alphabetical order) is dropped and the intercept becomes the mean of this level.

```
url1 <- "http://sma.epfl.ch/~lbelzile/math341/growth.dat"
growth <- read.table(url1, header = TRUE)
summary(growth)
```

```
##        y               x            group
##  Min.   :107.0   Min.   : 78.00   Length:20
##  1st Qu.:122.0   1st Qu.: 93.75   Class :character
##  Median :140.0   Median :100.50   Mode  :character
##  Mean   :142.4   Mean   :100.90
##  3rd Qu.:156.5   3rd Qu.:109.25
```

```
##  Max.   :185.0   Max.   :123.00
```

```
##Check what the factor encodes: transpose of design matrix
t(model.matrix(y ~ group - 1, data= growth))
```

```
##                 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
## groupcontrol    1 1 1 1 1 1 1 1 1  1  0  0  0  0  0  0  0  0  0  0
## groupthiouracil 0 0 0 0 0 0 0 0 0  0  1  1  1  1  1  1  1  1  1  1
## attr(,"assign")
## [1] 1 1
## attr(,"contrasts")
## attr(,"contrasts")$group
## [1] "contr.treatment"
```

```
#Fit linear model with interaction
rats_lm <- lm(y ~ x * group, data = growth)
## recall x*group is equivalent to x + group + x:group
## x:group is the interaction term,

## The design matrix can be extracted using the command
model.matrix(rats_lm)
```

```
##    (Intercept)   x groupthiouracil x:groupthiouracil
## 1            1 114                0                 0
## 2            1 123                0                 0
## 3            1 111                0                 0
## 4            1 100                0                 0
## 5            1 104                0                 0
## 6            1 102                0                 0
## 7            1  94                0                 0
## 8            1 112                0                 0
## 9            1  90                0                 0
## 10           1 110                0                 0
## 11           1 109                1               109
## 12           1 101                1               101
## 13           1 100                1               100
## 14           1 100                1               100
## 15           1 101                1               101
## 16           1  92                1                92
## 17           1  95                1                95
## 18           1  93                1                93
## 19           1  78                1                78
## 20           1  89                1                89
## attr(,"assign")
## [1] 0 1 2 3
## attr(,"contrasts")
## attr(,"contrasts")$group
## [1] "contr.treatment"
```

```
## 95% confidence interval
confint(rats_lm, level = 0.95)[3:4,]
```

```
##                       2.5 %      97.5 %
```

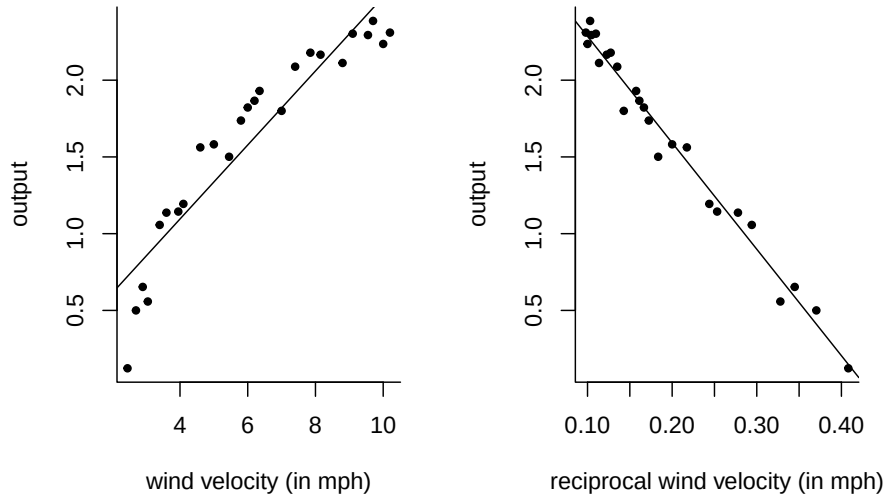
```
## groupthiouracil   -87.550757 134.1308431
## x:groupthiouracil  -1.594919   0.6083507
```

```
## Generalized linear hypothesis test for mu=gamma=0
## covered later in the course
#car::linearHypothesis(rats_lm, rbind(c(0,0,1,0), c(0,0,0,1)), c(0,0))
```

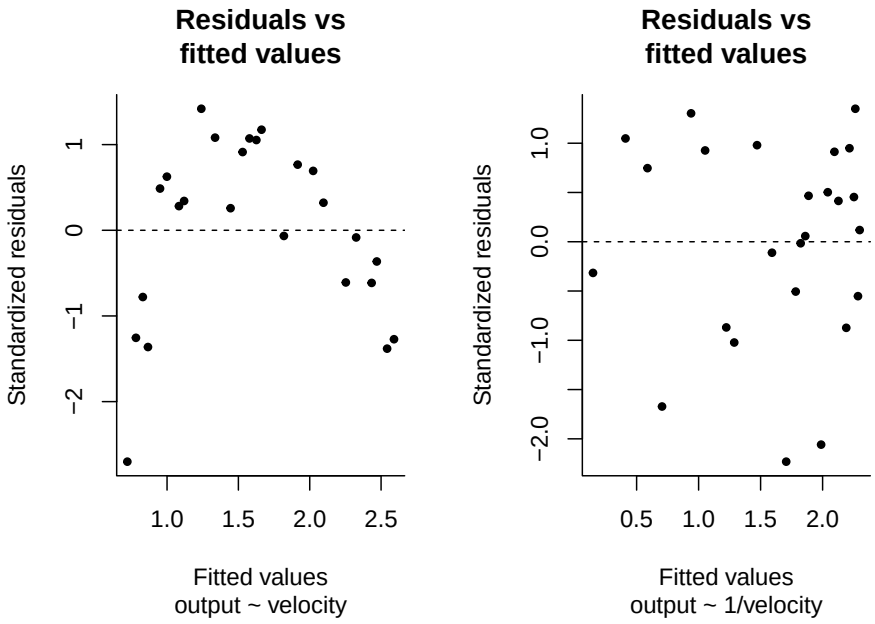
4.5.2 Exercise 7.2 - Electric production of windmills

The dataset `windmill` contains measurements of electricity output of wind turbine over 25 separate fifteen minute periods. We are interested in the relation between direct output and the average wind speed (measured in miles per hour) during the recording. a. Fit a linear model with wind speed as covariate and plot the standardized residuals against the fitted values. Do you notice any residual structure missed by the model mean specification? Try fitting a model using the reciprocal of wind speed as covariate. Comment on the adequacy of the models. b. Predict, using both models in term, the output of electricity given that the average wind speed in a given period is 5 miles per hour. Provide prediction interval for your estimates. c. Produce a standard Gaussian quantile-quantile plot of the standardized residuals. Superimpose approximate pointwise confidence intervals.

```
#Extract dataset
url2 <- "http://sma.epfl.ch/~lbelzile/math341/windmill.dat"
windmill <- read.table(file = url2, header = TRUE)
#Copy variables
output <- windmill$output
velocity <- windmill$velocity
recip_velo <- 1/velocity
#Fit linear model
lm_wind1 <- lm(output ~ velocity)
#Summary of fit
summ1 <- summary(lm_wind1)
#Graphical parameters
#mfrow: 1 line 2 column plotting window,
#pch: small dots plotting symbol,
#bty: L console shape)
par(mfrow = c(1, 2), pch = 20, bty = "l")
#Plot and add line of best fit
plot(y = output, x = velocity, xlab = "wind velocity (in mph)")
abline(lm_wind1)
#Repeat with second dataset
summ2 <- summary(lm_wind2 <- lm(output ~ recip_velo))
#alternatively summary(lm_wind2 <- lm(output ~ I(1/velocity)))
#Note above how we can assign variables inside call to other functions
plot(output ~ recip_velo, xlab = "reciprocal wind velocity (in mph)")
abline(lm_wind2)
```




```r
#Standardized residuals r - manual calculation
#Standard deviation of errors
s <- sqrt(sum(resid(lm_wind1)^2)/lm_wind1$df.residual)
#Design matrix i.e. cbind(1, velocity)
Xmat1 <- model.matrix(lm_wind1)
#Dimensions
n <- nrow(Xmat1)
p <- ncol(Xmat1)
#Projection matrix onto Xmat1
Hmat1 <- Xmat1 %*% solve(crossprod(Xmat1)) %*% t(Xmat1)
#Diagonal of H
leverage <- diag(Hmat1)
#Standardized residuals
r_wind1 <- resid(lm_wind1)/(s*sqrt(1-leverage))
#The function rstandard returns those for us
r_wind2 <- rstandard(lm_wind2)

#Plot of standardized residuals vs fitted values
plot(y = r_wind1 - mean(r_wind1), x = fitted(lm_wind1),
     ylab = "Standardized residuals", xlab = "Fitted values",
     main = "Residuals vs\nfitted values", sub ="output ~ velocity")
abline(h = 0, lty = 2)
plot(y = r_wind2 - mean(r_wind2), x = fitted(lm_wind2),
     ylab = "Standardized residuals", xlab = "Fitted values",
     main = "Residuals vs\nfitted values", sub ="output ~ 1/velocity")
abline(h = 0, lty = 2)
```



There is some structure left in the model `output ~ velocity`, since the smallest values occur at the endpoint of the output. There is less visible structure in the model with the reciprocal. The second model appears to fit better, since its R^2 value is 0.98 compared to 0.87 for the first model. Note that, in the second model, the intercept corresponds to infinite strenght wind gusts.

```
#Predict new observation
pred1int <- predict(lm_wind1, newdata = data.frame(velocity = 5),
                    interval = "prediction")
pred2int <- predict(lm_wind2, newdata = data.frame(recip_velo = 1/5),
                    interval = "prediction")
#Manually, see slide 68
xplus <- c(1, 5)
pred1 <- xplus %*% coef(lm_wind1)
interv_length <- qt(0.975, lm_wind1$df.residual) * summ1$sigma *
           sqrt((1 + t(xplus) %*% solve(crossprod(Xmat1)) %*% xplus))
#Check that the calculation is correct
isTRUE(all.equal(c(pred1, pred1 - interv_length, pred1 + interv_length),
                 c(pred1int), check.attributes = FALSE))
```

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

The predicted output is 1.34 units of electrity for the first model, while the point forecast is 1.59 for the model with the reciprocal velocity. Both intervals overlap, but the second one [1.39, 1.79] is considerable narrower than the first one, given by [0.84, 1.84].

```
summary(update(lm_wind1, . ~ .-1))
```

```
##
## Call:
## lm(formula = output ~ velocity - 1)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.51276 -0.17156  0.08675  0.20282  0.36832
```

```
##
## Coefficients:
##          Estimate Std. Error t value Pr(>|t|)
## velocity  0.25949    0.00715   36.29   <2e-16
##
## Residual standard error: 0.2364 on 24 degrees of freedom
## Multiple R-squared:  0.9821, Adjusted R-squared:  0.9814
## F-statistic:  1317 on 1 and 24 DF,  p-value: < 2.2e-16
```

The function `update` changes the arguments of the linear model. Here, the `.` means keep all variables on lhs or rhs. You can also use it with a dataset to fit all the remaining variables after specifying the response variable, like for example `lm(output ~ .,  data = windmill)` would have `velocity` as covariate.

We notice first that the confidence interval for β_0, the intercept, includes zero, we cannot reject the null hypothesis that $\beta_0 = 0$ at level 0.95.

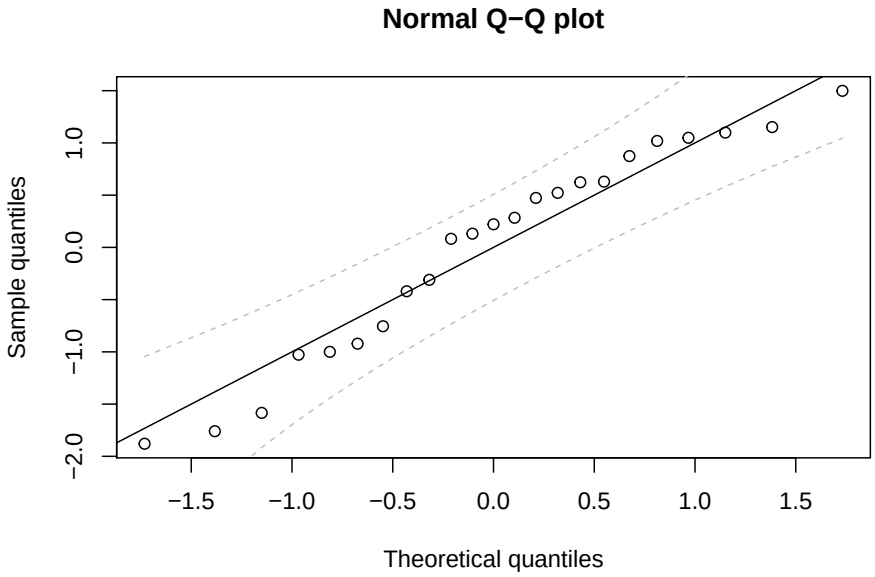
The coefficient β_1 corresponding to the effect of velocity has a smaller standard error than the first model. Does this make sense? If a model is correctly specified, addition of new variables that are unrelated does not introduce bias, but ncessarily inflates the standard errors by Gauss–Markov theorem. However, if the intercept should truly be there (this can be made necessary because of measurement errors) and $\beta_0 \neq 0$, then the tests and confidence intervals will be invalid in the simplified model.

The multiple R_c^2 goes up the roof, but makes no sense here because it compares two models that are not nested (the model with a single mean versus which has no constant). A consequence of the removal of the intercept is that the average of the residuals is not zero anymore and that R returns different values for the `Multiple R-squared`.

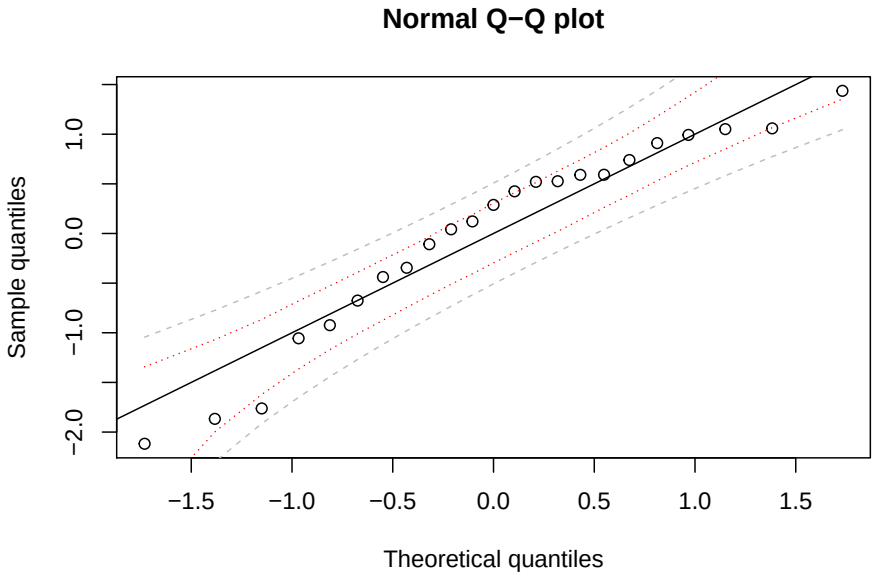
If you remove the intercept in a `lm` object using `-1`, the value returned by `summary` for the coefficient `Multiple R-squared` is the R^2, not R_c^2!

We now produce the quantile-quantile plots using the results described in Section 4.4.

```
Q <- t(qr.Q(qr(Xmat1), complete = TRUE))
resQ1 <- (t(Q) %*% resid(lm_wind1))[-(1:2)]
#Function to add confidence intervals using order statitics
confint.qqplot.ptw <- function(n, dist = "norm", ...){
  t(sapply(1:n, function(i){
  #Beta order statistic quantiles, mapped to scale dist
    do.call(paste0('q', dist), list(qbeta(c(0.025, 0.975), i, n - i + 1), ...))
  }))
}

# Adjust the number of observations
N <- n - p
#Plotting positions on X axis
rankit <- qnorm((1:N) / (N+1))
plot(rankit, sort(scale(resQ1)), xlab = "Theoretical quantiles",
    ylab = "Sample quantiles", main = "Normal Q-Q plot")
abline(a = 0, b = 1)
confint_ptwise <- confint.qqplot.ptw(N)
lines(rankit, confint_ptwise[,1], col = "gray", lty = 2)
lines(rankit, confint_ptwise[,2], col = "gray", lty = 2)
```




```
boot_samps <- replicate(sort(scale(rnorm(N))), n = (B <- 9999))
alpha <- 0.05
k <- alpha/2*(B + 1)
confint_boot <- apply(boot_samps, 1, sort)[c(k, B+1-k),]

#Example with second model
Xmat2 <- cbind(1, 1/velocity)
Q <- t(qr.Q(qr(Xmat2), complete = TRUE))
resQ2 <- (t(Q) %*% resid(lm_wind2))[-(1:2)]

plot(rankit, sort(scale(resQ2)), xlab = "Theoretical quantiles",
    ylab = "Sample quantiles", main = "Normal Q-Q plot")
abline(a = 0, b = 1)
confint_ptwise <- confint.qqplot.ptw(N)
#Simulated pointwise bands
lines(rankit, confint_boot[1,], col = "red", lty = 3)
lines(rankit, confint_boot[2,], col = "red", lty = 3)
#Theoretical bands based on order statistics distribution
lines(rankit, confint_ptwise[,1], col = "gray", lty = 2)
lines(rankit, confint_ptwise[,2], col = "gray", lty = 2)
```

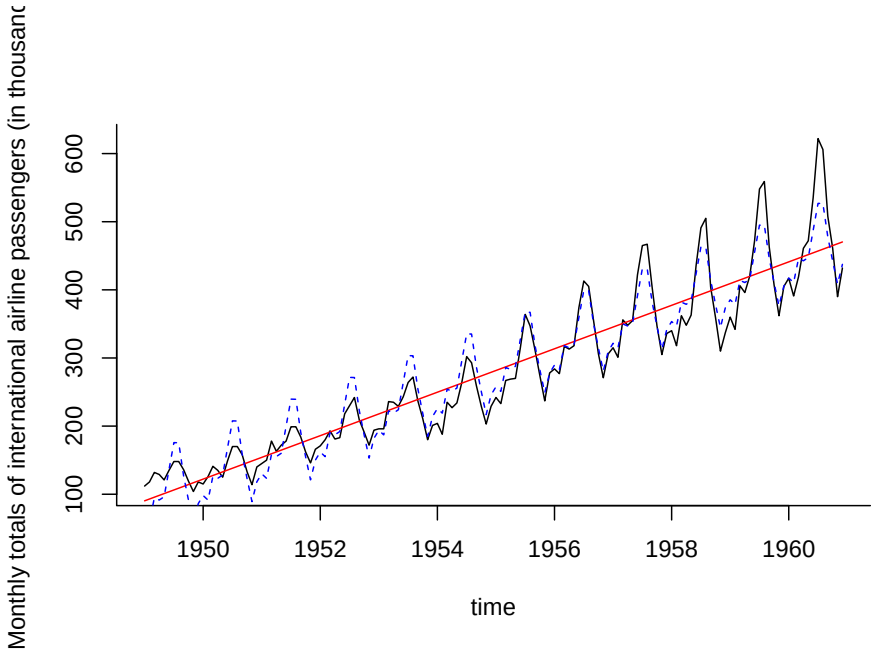


The simulated pointwise confidence interval are shorter and account for the scaling.

4.5.3 Exercise 7.3 - Air traffic

First load the data set and plot the observations

```
rm(list = ls()) #clear environment
par(bty = "l", pch = 20)
url3 <- "http://sma.epfl.ch/~lbelzile/math341/airpassengers.dat"
airpass <- read.table(file = url3, header = TRUE)
# Cast monthly binary to factor
airpass$time <- airpass$year + (airpass$month-1)/12
airpass$month <- as.factor(airpass$month)
attach(airpass)
#Proceed as usual
plot(y = passengers, x = time, type = "l",
     ylab = "Monthly totals of international airline passengers (in thousands)")
#Fit simple linear model with time as covariate
sum_ap <- summary(fit_ap <- lm(passengers ~ time))
lines(time, fitted(fit_ap), col = 2)
#Create monthly dummies
#create factor using `as.factor
month <- as.factor(rep(1:12, length = length(time)))
levels(month) <- month.abb #abbreviation of months
#A fancier way would convert the fraction to units,
#month <- as.factor(1 + as.integer(c(time*12) %% 12)) # %% is modulo operator
#quarter <- as.factor(rep(1:4, each = 3, length = length(time)))
sum_ad <- summary(fit_ad <- lm(passengers ~ time + month))
lines(time, fitted(fit_ad), lty = 2, col = 4) #dashed blue line
```




```
#Prediction
predict(fit_ad, newdata = data.frame(time = 1962+11/12, month = month[12]))
```

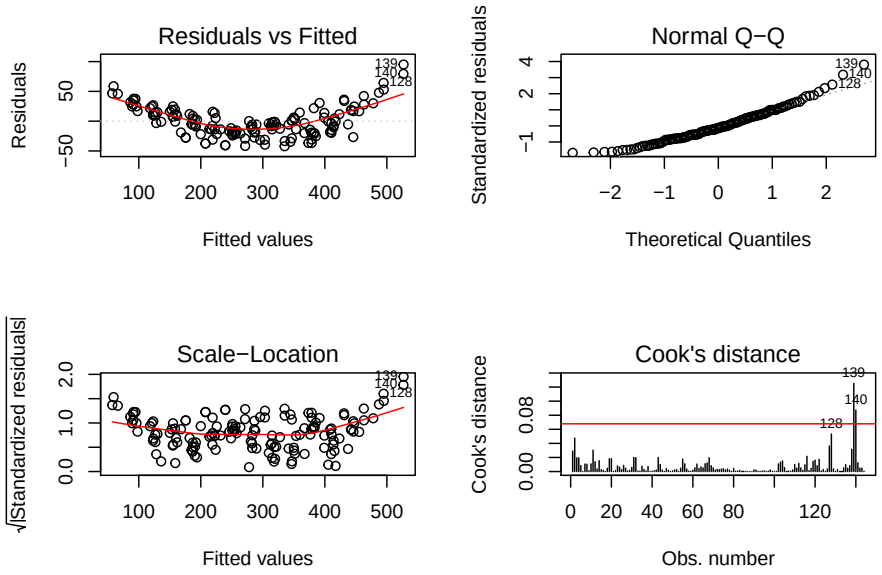
```
##       1
## 501.263
```

```
coef(fit_ad) %*% c(1, 1962 + 11/12, rep(0, 10), 1) #baseline is January if global mean included
```

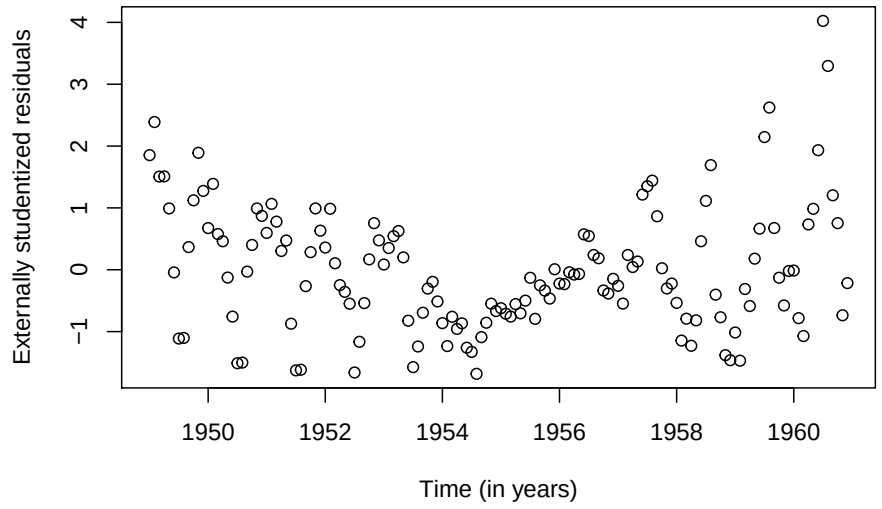
```
##          [,1]
## [1,] 501.263
```

We notice that the model does an overall good job at getting the big features, but misses many things. The first point is that the relationship is not quite linear: a residual plot shows a somewhat quadratic relation between the fitted values and the residuals. The second obvious feature not captured is the change in the variation (the amplitude of the wave pattern changes over time). Since the variance is increasing, a log-transformation may help stabilize it. The residuals are not apparently close to normal (the last values are systematically too large) and there is some skewness. The last few points have large leverage and drive the curve up.

```
n <- length(passengers)
p <- length(coef(fit_ad))
par(mfrow = c(2, 2))
plot(fit_ad, which = 1) #residuals vs fitted values
plot(fit_ad, which = 2) #Normal Q-Q plot
plot(fit_ad, which = 3) #standardized residuals vs fitted values
plot(fit_ad, which = 4, sub.caption = "") #Cook distance plot
abline(h = 8/(n - 2*p), col = 2)
```

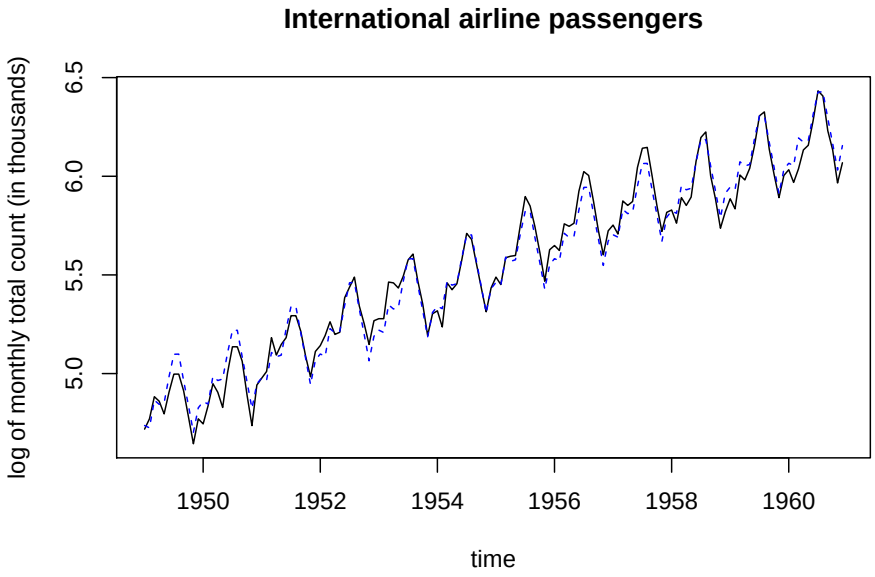



```
par(mfrow = c(1, 1)) #return to one plot per window
#Compute Cook statistic and other influence statistics
infl_ad <- influence.measures(fit_ad)
cookval_ad <- infl_ad$infmat[,"cook.d"] #cooks.distance
#Diagonal values of the "hat" projection matrix
h_ad <- infl_ad$infmat[, "hat"] #hatvalues
plot(time, rstudent(fit_ad),
     ylab = "Externally studentized residuals",
     xlab = "Time (in years)")
```

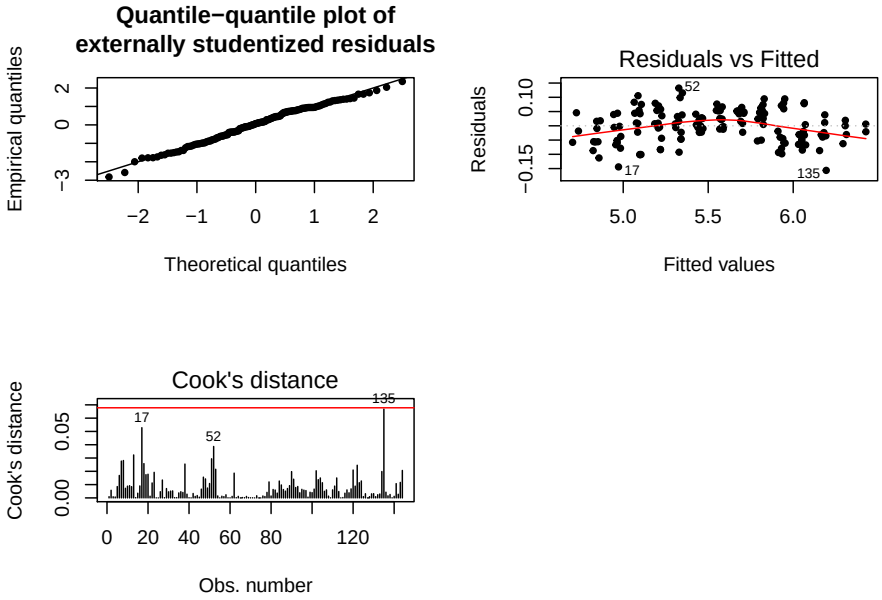


Let us consider the log counts. The fit is much better and the quadratic relationship with the residuals vs fitted values is attenuated. While some points still have high leverage value, they are not considered outliers.

```
fit_l <- lm(log(passengers) ~ time + month)
plot(log(passengers) ~ time, main = "International airline passengers",
ylab = "log of monthly total count (in thousands)", type = "l")
lines(time, fitted(fit_l), lty = 2, col = 4)
```



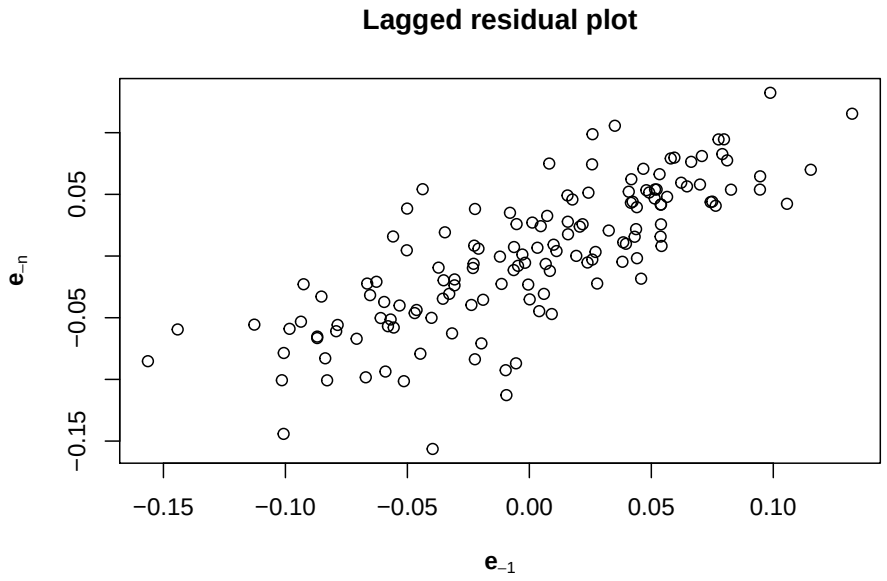

```
par(mfrow = c(2, 2), pch = 20)
# Q-Q plot
plot(x = qt((1:n)/(n+1), df = n - p + 1), y = sort(scale(rstudent(fit_l))),
     xlab = "Theoretical quantiles", ylab = "Empirical quantiles",
     main = "Quantile-quantile plot of\nexternally studentized residuals")
abline(a = 0, b = 1)
plot(fit_l, which = 1, sub.caption = "")
plot(fit_l, which = 4, sub.caption = "")
abline(h = 8/(n - 2*p), col = 2)
```



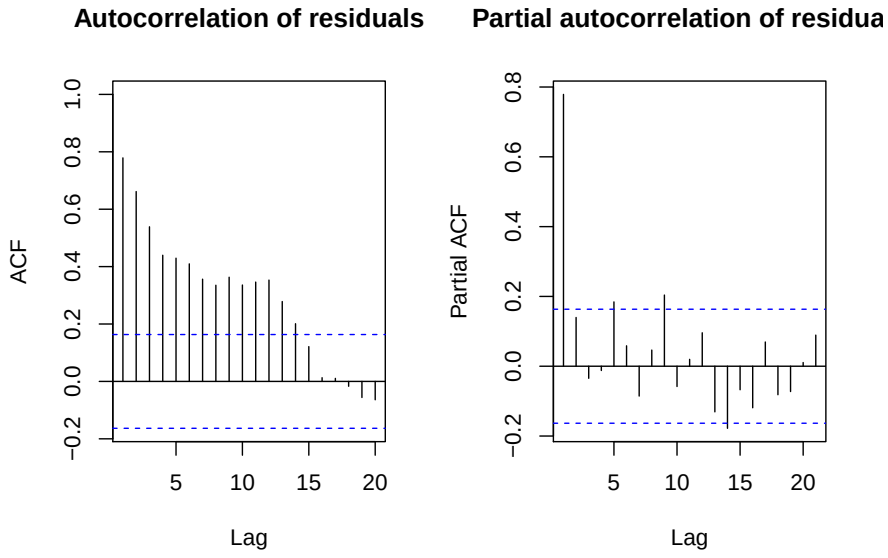
One hypothesis of the linear model that is clearly violated here is the independence of the errors. Even if the variance $\mathrm{Var}(\boldsymbol{e}) = \sigma^2 \mathbf{M}_{\mathbf{X}}$ need not have independent errors, there is positive dependence from residual to residual. This is confirmed by looking at the autocorrelation, which indicates geometric decay. This will be covered in MATH 342 (Time Series), but you should just think here of shocks carrying through until the next period before the model reverts to the mean.

Ignoring the serial dependence in the error has consequences: the standard errors are too small (since errors are correlated, there is less units of information so we are overconfident in our uncertainty quantification).

```
#Laggedresidual plots
par(mfrow = c(1, 1))
plot(x = resid(fit_l)[-1], y = resid(fit_l)[-n],
  ylab = expression(bold(e)[-n]), xlab = expression(bold(e)[-1]),
  main = "Lagged residual plot")
```




```
#(partial) correlogram: if residuals have no structure
#there should not be anything outside the bands 19 times out of 20
#Covered in detail in MATH-342 (Time series)
par(mfrow = c(1, 2))
acf(resid(fit_l), main = "Autocorrelation of residuals", xlim = c(1,20))
pacf(resid(fit_l), main = "Partial autocorrelation of residuals")
```



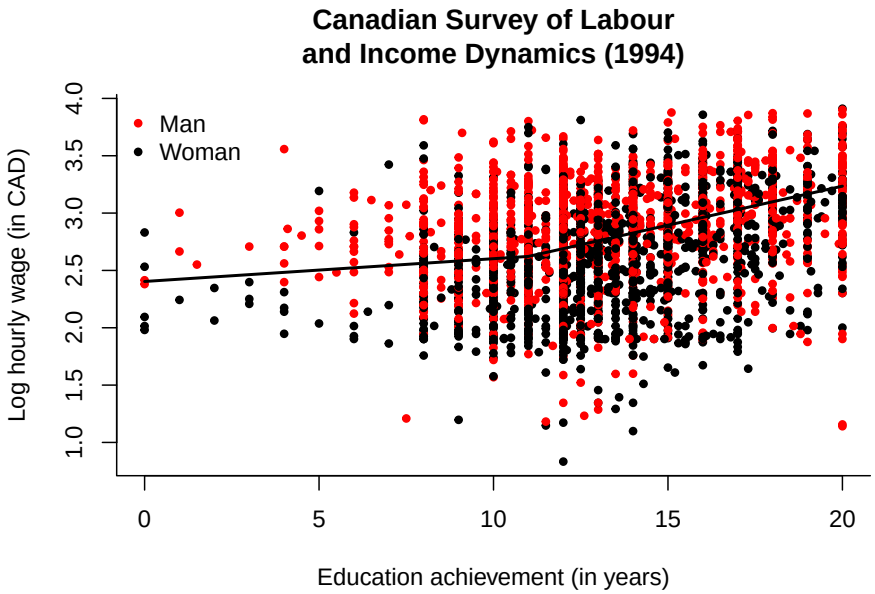
```
#detach dataset
detach(airpass)
```

4.5.4 Exercise 7.4 - Determinants of earnings

```
url4 <- "http://sma.epfl.ch/~lbelzile/math341/labour.dat"
labour <- read.table(url4, header = TRUE, stringsAsFactors = TRUE)
attach(labour)
## Create dummy for extract columns
## additional years of schooling after high school
labour$pseduc  <- I(education >= 13) * (education - 13)
educ_lm <- lm(lhwages ~ education + pseduc, data = labour)
confint(educ_lm)
```

```
##                 2.5 %     97.5 %
## (Intercept) 2.29267414 2.51468758
## education   0.01030330 0.02940631
## pseduc      0.03325994 0.06318974
```

```
## Plot with meaningful title + axis label
## red for Male, black for Female
plot(lhwages ~ education, pch = 20, col = as.integer(gender),
     bty = "l", ylab = "Log hourly wage (in CAD)",
     xlab = "Education achievement (in years)",
      main = "Canadian Survey of Labour\nand Income Dynamics (1994)")
## Create observations on a grid, reproducing the design
predic <- cbind(1, 0:20, c(rep(0,12), 1:9)) %*% coef(educ_lm)
lines(0:20, predic, lwd = 2)
## Legend
legend(x = "topleft", legend = c("Man","Woman"),
       col = c(2, 1), pch = 20, bty = "n")
```



```
## Clear pattern: women are paid less for equivalent qualification

## Add gender as covariate
educ_lm <- lm(lhwages ~ ., data= labour)
# fit lm with all columns but lhwages
## or equivalently
#update(educ_lm, . ~ . + gender)
summary(educ_lm)
```

```
##
## Call:
## lm(formula = lhwages ~ ., data = labour)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.10953 -0.25249  0.02989  0.28176  1.27908
##
## Coefficients:
##             Estimate Std. Error t value     Pr(>|t|)
## (Intercept) 2.200103   0.055375  39.731      < 2e-16
## education   0.026589   0.004673   5.690 0.0000000138
## genderMale  0.256471   0.014771  17.363      < 2e-16
## pseduc      0.037459   0.007322   5.116 0.0000003308
##
## Residual standard error: 0.4154 on 3183 degrees of freedom
## Multiple R-squared:  0.1897, Adjusted R-squared:  0.1889
## F-statistic: 248.3 on 3 and 3183 DF,  p-value: < 2.2e-16
```

```
confint(educ_lm)
```

```
##                  2.5 %     97.5 %
## (Intercept) 2.09152815 2.30867726
## education   0.01742684 0.03575089
## genderMale  0.22750913 0.28543379
## pseduc      0.02310256 0.05181598
```

```
detach(labour)
```

The coefficient for gender is still statistically significative at level $\alpha = 5\%$ after adjusting for the education level.

Chapter 5

Analysis of variance

Consider the linear model $\boldsymbol{Y} = \mathbf{1}_n\alpha + \mathbf{Z}\boldsymbol{\beta} + \boldsymbol{\varepsilon}$ where $\mathbf{X} = (\mathbf{1}_n^\top, \mathbf{Z}^\top)^\top$ is a full rank $n \times p$ design matrix. Let as usual $\text{TSS} = \boldsymbol{y}^\top \mathbf{M}_{\mathbf{1}_n}\boldsymbol{y}$, the total sum of square, and $\text{RSS} = \boldsymbol{y}^\top \mathbf{M}_{\mathbf{X}}\boldsymbol{y}$, the sum of squared residuals. Under the assumptions of the Gaussian linear model (or asymptotically), the F-test statistic for testing the null hypothesis $\text{H}_0 : \boldsymbol{\beta} = \mathbf{0}_{p-1}$ against the alternative $\text{H}_a : \boldsymbol{\beta} \in \mathbb{R}^{p-1}$ assuming the larger model is correctly specified is

$$F = \frac{(\text{TSS} - \text{RSS})/(p-1)}{\text{RSS}/(n-p)}.$$

Under the null hypothesis, $F \sim \mathcal{F}(p-1, n-p)$.

An ANOVA table (`anova`) arranges the information about the sum of squares decomposition, the degree of freedom and the value of the F test statistic in the following manner.

Sum of squares	degrees of freedom	scaled sum of squares	test statistic	P-value
ESS	$p-1$	$\text{ESS}/(p-1)$	F	$1 - \texttt{pf}(F, p-1, n-p)$
RSS	$n-p$	$\text{RSS}/(n-p)$		

5.1 Sum of squares decomposition

Consider the orthogonal decomposition

$$\boldsymbol{y}^\top \boldsymbol{y} = \boldsymbol{y}^\top \mathbf{M}_{\mathbf{X}}\boldsymbol{y} + \boldsymbol{y}^\top \mathbf{H}_{\mathbf{X}}\boldsymbol{y},$$

along with the model

$$\boldsymbol{y} = \beta_0 \mathbf{1}_n + \mathbf{X}_1\boldsymbol{\beta}_1 + \mathbf{X}_2\boldsymbol{\beta}_2 + \boldsymbol{\varepsilon}.$$

We consider four concurrent models, for $\mathbf{X}_1$ an $n \times p_1$ matrix and $\mathbf{X}_2$ and $n \times p_2$ matrix and $\mathbf{X}_a = (\mathbf{1}_n^\top, \mathbf{X}_1^\top, \mathbf{X}_2^\top)^\top$ and $n \times p = n \times (1 + p_1 + p_2)$ full rank matrix.

(a) the full model with both predictors, $\text{M}_a : \boldsymbol{y} = \beta_0\mathbf{1}_n + \mathbf{X}_1\boldsymbol{\beta}_1 + \mathbf{X}_2\boldsymbol{\beta}_2 + \boldsymbol{\varepsilon}.$,
(b) the restricted model with $\text{M}_b : \boldsymbol{\beta}_2 = 0$ and only the first predictor, of the form $\boldsymbol{y} = \beta_0\mathbf{1}_n + \mathbf{X}_1\boldsymbol{\beta}_1 + \boldsymbol{\varepsilon}.$,
(c) the restricted model with $\text{M}_c : \boldsymbol{\beta}_1 = 0$ and only the second predictor, of the form $\boldsymbol{y} = \beta_0\mathbf{1}_n + \mathbf{X}_2\boldsymbol{\beta}_2 + \boldsymbol{\varepsilon}$.
(d) the intercept-only model $\text{M}_d : \boldsymbol{y} = \beta_0\mathbf{1}_n + \boldsymbol{\varepsilon}$.

Let $\mathbf{X}_a$, $\mathbf{X}_b$, $\mathbf{X}_c$, and $\mathbf{X}_d$ be the corresponding design matrices.

R uses an orthogonal decomposition of the projection matrix on to $\mathbf{X}_a$, $\mathbf{H}_{\mathbf{X}_a}$ into two parts: $\mathbf{H}_{\mathbf{X}_a} = \mathbf{H}_{\mathbf{X}_b} + \mathbf{H}_{\mathbf{M}_{\mathbf{X}_b}\mathbf{X}_2}$. The last term is the contribution of $\mathbf{X}_2$ to the model fit when $\mathbf{1}_n, \mathbf{X}_1$ are already part of the model. We can form the sum of squares of the regression using this decomposition. We use the notation $\text{SSR}(\mathbf{H}) = \boldsymbol{y}^\top \mathbf{H}\boldsymbol{y}$ to denote the sum of squares obtained by projecting $\boldsymbol{y}$ onto the span of $\mathbf{H}$.

We have

$$\text{SSR}(\mathbf{H}_{\mathbf{M}_{\mathbf{X}_b}\mathbf{X}_2}) = \text{SSR}(\mathbf{H}_{\mathbf{X}_a}) - \text{SSR}(\mathbf{H}_{\mathbf{X}_b}).$$

that is, the difference in sum of squared from the regression with model M_a versus that from regression M_b. This is the sum of squares from the regression that are due to the addition of $\mathbf{X}_2$ to a model that already contains $(\mathbf{1}_n, \mathbf{X}_1)$ as regressors.

The usual F-test statistic for the null hypothesis $\mathcal{H}_0 : \boldsymbol{\beta}_2 = 0$ can be written as

$$F = \frac{\text{SSR}(\mathbf{H}_{\mathbf{M}_{\mathbf{X}_b}\mathbf{X}_2})/p_2}{\text{RSS}_a/(n-p)} = \frac{(\text{RSS}_b - \text{RSS}_a)/p_2}{\text{RSS}_a/(n-p)} \sim \mathcal{F}(p_2, n-p).$$

The last equality follows by noting that $\text{SSR}(\mathbf{H}_{\mathbf{X}_b}) + \text{RSS}_b = \text{SSR}(\mathbf{H}_{\mathbf{X}_a}) + \text{RSS}_a$.

5.1.1 The decomposition of squares in R

Let us illustrate how to obtain the various quantities presented above using the **R** outputs.

First, we look at some data. The dataset `Chirot` from the package `carData` contains information about the 1907 Romanian peasant rebellion. We model the intensity of the rebellion as a function of commercialization of agriculture and a measure of traditionalism. We start by fitting the four models $\text{M}_a, \text{M}_b, \text{M}_c, \text{M}_d$ detailed above with the regressors $\mathbf{X}_1 \equiv$`commerce`, $\mathbf{X}_2 \equiv$`tradition` and an intercept.

```
#install.packages("carData")
data(Chirot, package = "carData")
## Fit linear model with commerce and tradition as explanatory variables
mod.a <- lm(intensity ~ commerce + tradition, data = Chirot)
mod.b <- update(mod.a, .~. - tradition)  #remove tradition
## mod.b is equivalent to
# mod.b <- lm(intensity ~ commerce, data = Chirot)
mod.c <- update(mod.a, .~. - commerce)  #remove tradition
mod.d <- lm(intensity ~ 1, data = Chirot)
```

First, the RSS from model M_a can be extracted from the table returned by `summary` under the label `Residual standard error:`. This gives $\widehat{\sigma}$, and $\text{RSS}_d = (n-p)\widehat{\sigma}$, where $n - p = 29$ in the present setting.

```
RSS.a <- crossprod(resid(mod.a))
RSS.a[1,1]
```

```
## [1] 41.43137
```

```
all.equal(c(RSS.a), summary(mod.a)$df[2] * summary(mod.a)$sigma^2)
```

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

The function `anova` outputs the following:

```
anova(mod.a)
```

```
## Analysis of Variance Table
##
## Response: intensity
##           Df Sum Sq Mean Sq F value      Pr(>F)
## commerce   1 50.066  50.066 35.0438 0.000001985
```

```
## tradition  1  6.074   6.074  4.2514     0.04828
## Residuals 29 41.431   1.429
```

The function `anova` considers the *sequential* decomposition $\mathbf{H}_{\mathbf{X}_a} = \mathbf{H}_{\mathbf{1}_n} + \mathbf{H}_{\mathbf{M}_{\mathbf{1}_n}\mathbf{X}_1} + \mathbf{H}_{\mathbf{M}_{\mathbf{X}_b}\mathbf{X}_2}$. The column `Sum Sq` gives

- 1st line: the contribution for `commerce`, $\boldsymbol{y}^\top \mathbf{H}_{\mathbf{M}_{\mathbf{1}_n}\mathbf{X}_1} \boldsymbol{y}$,
- 2nd line: $\boldsymbol{y}^\top \mathbf{H}_{\mathbf{M}_{\mathbf{X}_b}\mathbf{X}_2} \boldsymbol{y}$ and
- 3rd line: the residuals sum of squares RSS_a.

These are the conditional sum of squares from the regression for the additional variable. The test statistics corresponding to the F and P-values in the table are

$$F_1 = \frac{\text{SSR}(\mathbf{H}_{\mathbf{M}_{\mathbf{1}_n}\mathbf{X}_1})/p_1}{\text{RSS}/(n-p)}$$

and

$$F_2 = \frac{\text{SSR}(\mathbf{H}_{\mathbf{M}_{\mathbf{X}_b}\mathbf{X}_2})/p_2}{\text{RSS}/(n-p)}.$$

Note that the residual sum of squares from the denominator is that of the full model in both cases. It is orthogonal to the numerator, but not equal to the residuals from the model M_b for F_1.

Recall that the order in which the variables enter the model matters when performing model selection unless your regressors are orthogonal. We thus obtain a different output if we specify instead

```
anova(lm(intensity ~ tradition + commerce, data = Chirot))
```

```
## Analysis of Variance Table
##
## Response: intensity
##           Df Sum Sq Mean Sq F value     Pr(>F)
## tradition  1 19.673  19.673  13.770   0.000872
## commerce   1 36.467  36.467  25.525 0.00002194
## Residuals 29 41.431   1.429
```

The `F` and `Pr(>F)` columns now correspond to

$$F_1' = \frac{\text{SSR}(\mathbf{H}_{\mathbf{M}_{\mathbf{1}_n}\mathbf{X}_2})/p_2}{\text{RSS}/(n-p)}$$

and

$$F_2' = \frac{\text{SSR}(\mathbf{H}_{\mathbf{M}_{\mathbf{X}_c}\mathbf{X}_1})/p_1}{\text{RSS}/(n-p)}.$$

5.1.2 Dropping or adding variables

The function `drop1` allows you to test for model simplification, the hypothesis that either model (b) or model (c) is an adequate simplification of the full model (a). The output includes the RSS value in addition to the sum of squared decomposition from the previous tables. In both cases here, the null hypothesis that the simpler model with $\beta_2 = 0$ or $\beta_1 = 0$ (against the alternative that the model with $\boldsymbol{\beta} \in \mathbb{R}^2$ is correct) is rejected at significance level $\alpha = 5\%$.

```
drop1(mod.a, test = 'F')
```

```
## Single term deletions
##
## Model:
## intensity ~ commerce + tradition
##           Df Sum of Sq    RSS    AIC F value     Pr(>F)
## <none>                 41.431 14.266
## commerce   1    36.467 77.898 32.469 25.5252 0.00002194
## tradition  1     6.074 47.505 16.643  4.2514    0.04828
```

These F values are the same as those obtained with the call on `anova` for the full model and the output is probably less confusing.

There is a similar command to add variables, called `add1`. You can try running `add1(mod.c, scope = .~. + tradition, test = 'F')` to obtain similar output to the `anova` call.

To test for a simplified model in which many of the variables are removed, we can use the general linear hypothesis framework. The function `glht` in the package `multcomp` handles this, as does the function `linearHypothesis` in `car`.

Note that in general, multiple testing leads to inflated Type-I error for the set of tests, meaning that the probability of rejecting at least one null hypothesis for m tests provided that they are all true is greater than the significance level α of the individual tests. A Bonferroni correction (take α/m as level if you perform m tests) could be made to alleviate this, but the power to detect will be lower.

```
#install packages `multcomp` and `car` if necessary
#Test both hypothesis separately with Bonferroni correction
jt_test <- multcomp::glht(mod.a, linfct = rbind(c(0, 1, 0), c(0, 0, 1)),
                          test = adjusted("bonf"))
summary(jt_test)
```

```
##
##   Simultaneous Tests for General Linear Hypotheses
##
## Fit: lm(formula = intensity ~ commerce + tradition, data = Chirot)
##
## Linear Hypotheses:
##        Estimate Std. Error t value  Pr(>|t|)
## 1 == 0  0.09522    0.01885   5.052 0.0000437
## 2 == 0  0.11992    0.05816   2.062    0.0911
## (Adjusted p values reported -- single-step method)
```

```
summary(jt_test, multcomp::Ftest())
```

```
##
##   General Linear Hypotheses
##
## Linear Hypotheses:
##        Estimate
## 1 == 0  0.09522
## 2 == 0  0.11992
##
## Global Test:
##       F DF1 DF2      Pr(>F)
## 1 19.65   2  29 0.000004038
```

```
#Test hypothesis jointly using GLHT
car::linearHypothesis(mod.a, hypothesis.matrix = rbind(c(0, 1, 0), c(0, 0, 1)), test = "F")
```

```
## Linear hypothesis test
##
## Hypothesis:
## commerce = 0
## tradition = 0
##
## Model 1: restricted model
## Model 2: intensity ~ commerce + tradition
##
##   Res.Df    RSS Df Sum of Sq      F      Pr(>F)
## 1     31 97.571
## 2     29 41.431  2     56.14 19.648 0.000004038
```

You can also use the function ANOVA to test with nested model. The syntax is slightly different, but the output is exactly the same.

```
#Simpler to more complex nested models
anova(mod.d, mod.a)
```

```
## Analysis of Variance Table
##
## Model 1: intensity ~ 1
## Model 2: intensity ~ commerce + tradition
##   Res.Df    RSS Df Sum of Sq      F      Pr(>F)
## 1     31 97.571
## 2     29 41.431  2     56.14 19.648 0.000004038
```

At this point, it is important to note that we will get different test statistics (due to different denominator) if we consider the residuals sum of squares (RSS) from the full model or the simplified model in the F-test. The test is sensitive to model misspecification and we must make sure the residual sum of squares in the denominator is at least consistent.

5.1.3 Biased estimators of the residual sum of square

There are two possible scenarios:

1. Underfitting: you omit a variable that should be present in the model (misspecified model).

Omitting relevant variables undully inflates the residual sum of squares. Indeed, if the true model is M_a with $\beta_2 \neq 0$, but that we fit model M_b of the form $\boldsymbol{y} = \beta_0 \mathbf{1}_n + \mathbf{X}_1\boldsymbol{\beta}_1 + \boldsymbol{\varepsilon}$, then the residuals sum of squares we obtain will be $\mathrm{RSS}_a + \mathrm{SSR}(\mathbf{H}_{\mathbf{M}_{\mathbf{X}_b}\mathbf{X}_2})$. This reduces the statistical significance of the other variables in turn because the F-statistic is pulled toward zero. Since our null hypothesis is that the simpler model is adequate, our power to reject the null is smaller and we will go for much simpler models than should be used.

2. Overfitting: suppose on the contrary that we use a bigger model M_a with spurious variables (overfit) and that the true model is M_b. The parameter estimate $\hat{\beta}_2$ should have expectation zero and, as a result, the additional decrease in the sum of squared residuals should be also zero for the redundant variable conditional on the rest. The estimator is still unbiased; the only difference is that we use up additional degrees of freedom in the test.

This is best illustrated using a little simulation:

```
set.seed(1234) #RNG sequence - makes output reproducible
x1 <- rnorm(100)
x2 <- rexp(100)
x3 <- rbinom(n = 100, size = 100, p = 0.2)
x4 <- runif(100)
y <- x1 + x2 + x3 + rnorm(100, sd = 4)
#RSS/(n-p)
a_u <- anova(lm(y ~ x1 + x2)) #underfit
a_c  <- anova(lm(y ~ x1 + x2 + x3)) #correct
a_o <- anova(lm(y ~ x1 + x2 + x3 + x4 )) #overfit

print(c("Underfit" = a_u['Residuals','Mean Sq'],
         "Correct" = a_c['Residuals','Mean Sq'],
        "Overfit" = a_o['Residuals','Mean Sq']
))
```

```
## Underfit  Correct  Overfit
## 34.39788 19.77853 19.96968
```

```
#What is Underfit sum of square?
(a_c['Residuals','Sum Sq'] + a_c['x3','Sum Sq'])/(a_c['Residuals','Df'] + 1)
```

```
## [1] 34.39788
```

If there are no interactions and you wish to compare main effects conditional on all the others main effects present in the model for each of the explanatory variables, you can use the function `Anova` from the package `car`. This is the so-called Type II Anova decomposition. You could retrieve the output directly from repeated calls to `anova`.

5.2 One-way ANOVA

A one-way ANOVA is a test for equality of means in different subpopulations. Under the assumption that the observations have a common variance and that they are normally distributed, this corresponds to a Gaussian linear model with binary indicators (factors) as explanatories. Suppose there are L possible levels and n_j is the number of observations for group $j = 1, \ldots, L$.

The one-way ANOVA model can be written as

$$y_{i_j,j} = \alpha_j + \varepsilon_{i_j,j}, \quad \varepsilon_{i_j,j} \overset{\text{iid}}{\sim} \mathcal{N}(0, \sigma^2) \qquad (j = 1, \ldots, L, i_j = 1, \ldots, n_j).$$

Let $\boldsymbol{y}_j = (y_{1,j}, \ldots, y_{n_j,j})^\top$ denote the vector of observations for the first group and similarly for $\boldsymbol{\varepsilon_j}$; we can stack observations into a single regression with a matrix $\mathbf{X}$ of indicators variables, viz.

$$\begin{pmatrix} \boldsymbol{y}_1 \\ \boldsymbol{y}_2 \\ \vdots \\ \boldsymbol{y}_L \end{pmatrix} = \begin{pmatrix} \mathbf{1}_{n_1} & \mathbf{0}_{n_1} & \cdots & \mathbf{0}_{n_1} \\ \mathbf{0}_{n_2} & \mathbf{1}_{n_2} & \ddots & \mathbf{0}_{n_2} \\ \vdots & \ddots & \ddots & \vdots \\ \mathbf{0}_{n_L} & \mathbf{0}_{n_L} & \cdots & \mathbf{1}_{n_L} \end{pmatrix} \begin{pmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_L \end{pmatrix} + \begin{pmatrix} \boldsymbol{\varepsilon}_1 \\ \boldsymbol{\varepsilon}_2 \\ \vdots \\ \boldsymbol{\varepsilon}_L \end{pmatrix}.$$

In a **balanced design**, the number of observations m is the same for each of the L levels of the factor. We can then write $\mathbf{X}^\top\mathbf{X} = m\mathbf{I}_L$ and the standard errors are the same.

To test $\mathrm{H}_0 : \alpha_1 = \cdots = \alpha_L$, we can use the usual sum of squares decomposition. The F-statistic for this test is

$$F = \frac{(\boldsymbol{y}^\top \mathbf{M}_{\mathbf{1}_n}\boldsymbol{y} - \boldsymbol{y}^\top \mathbf{M}_{\mathbf{X}}\boldsymbol{y})/(L-1)}{\boldsymbol{y}^\top \mathbf{M}_{\mathbf{X}}\boldsymbol{y}/(n-L)} \sim \mathcal{F}(L-1, n-L).$$

and we reject the null hypothesis $\mu_1 = \cdots = \mu_L$ against the alternative that at least one group mean is different. One crucial assumption is that all groups have the same variance.

5.3 Two-way ANOVA and irrelevant hypotheses

A two-way analysis extends the one-way ANOVA by considering two factors $\mathbf{D}_1$ and $\mathbf{D}_2$ respectively taking L_1 and L_2 levels. The full model with interactions is usually specified using the design matrix $\mathbf{X} = [\mathbf{1}_n, \mathbf{D}_1, \mathbf{D}_2, (\mathbf{D}_1 \circ \mathbf{D}_2)]$, so the mean model is

$$\mathrm{E}(Y_i) = \omega + \sum_{j=1}^{L_1-1} \beta_j \mathbf{1}_{D_{i1}=j+1} + \sum_{k=1}^{L_2-1} \gamma_k \mathbf{1}_{D_{i2}=k+1} + \sum_{j=1}^{L_1-1}\sum_{k=1}^{L_2-1} \nu_{jk} \mathbf{1}_{D_{i1}=j+1}\mathbf{1}_{D_{i2}=k+1}.$$

Different submodels can be specified:

i. the intercept-only model, M_1, has 1 parameter ($\nu_{jk} = \beta_j = \gamma_k = 0, j = 1, \ldots, L_1 - 1, k = 1, \ldots, L_2 - 1$)
ii. the model M_2 with only the second factor ($\nu_{jk} = \beta_j = 0, j = 1, \ldots, L_1 - 1, k = 1, \ldots, L_2 - 1$) has L_2 parameters
iii. the model M_3 with only the first factor ($\nu_{jk} = \gamma_k = 0, j = 1, \ldots, L_1 - 1, k = 1, \ldots, L_2 - 1$) has L_1 parameters
iv. the additive model M_4 ($\nu_{jk} = 0, j = 1, \ldots, L_1 - 1, k = 1, \ldots, L_2 - 1$), has $L_1 + L_2 - 1$ parameters;
v. the additive model M_5 with the interaction term has $L_1 L_2$ parameters;

Note that we consider interaction terms only if the corresponding main effects are included. Removing main effects or lower-order interactions while keeping higher order terms implies that the levels of the factor are not arbitrarily labelled and rearrangement of the factors (changing the baseline) leads to potentially different conclusion.

To see this, consider the model with design matrix $[\mathbf{1}_n, \mathbf{D}_1, (\mathbf{D}_1 \circ \mathbf{D}_2)]$, so $\gamma_k = 0$ ($k = 1, \ldots, L_2 - 1$).

The mean level for observations in the different groups are

$$\begin{aligned}
\mathrm{E}(Y_i \mid D_{i1} = 1, D_{i2} = k) &= \omega \\
\mathrm{E}(Y_i \mid D_{i1} = j, D_{i2} = 1; j > 1) &= \omega + \beta_j \\
\mathrm{E}(Y_i \mid D_{i1} = j, D_{i2} = k; j, k > 1) &= \omega + \beta_j + \nu_{jk}
\end{aligned}$$

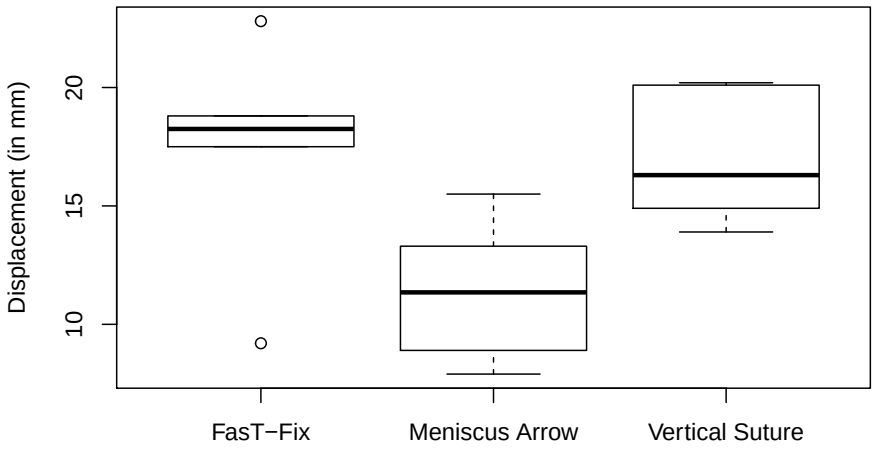
The effect of moving from level $D_2 = 1$ to $D_2 = k > 1$ is zero if $D_1 = 1$, whereas it is ν_{jk} if $D_1 > 1$. We therefore are treating the levels as being different and the models fitted (and the inference) will differ if we change the baseline category for the factors. This means that the latter is no longer arbitrary! In **R** this baseline is the first factor in alphabetical order, so is essentially arbitrary. In the same way we will not consider interactions without main effects, one would not consider polynomials of degree k without including the coefficients of the lower order. For a degree two polynomial of the form $\sum_{i=0}^{k} \beta_i x^i$, testing the hypothesis $\beta_i = 0$ for $i < k$ means restricting attention to polynomials whose kth order term is zero, which is usually nonsensical. Thus, any Wald test (the t values output by `summary`) corresponding to such an hypothesis should be disregarded.

5.4 Solutions

5.4.1 Exercise 9.3 - One-way analysis of variance

In this example, the three groups have very different variances! We carry out the testing procedure simply to show you how it is done (this does not imply we should use it in practice).

```
options(show.signif.stars = FALSE)
#http://www.stat.ufl.edu/~winner/data/meniscus.txt
meniscus <- read.table("http://sma.epfl.ch/~lbelzile/math341/meniscus.dat", header = TRUE)
boxplot(displacement ~ method, data = meniscus, ylab = "Displacement (in mm)")
```




```
lmod_men <- lm(displacement ~ method, data = meniscus)
summ_men <- summary(lmod_men)
#ANOVA table
anov_men <- anova(lmod_men)
#F statistic with dof
summ_men$fstatistic
```

```
##     value     numdf     dendf
##  5.783689  2.000000 15.000000
```

```
#P-value
anov_men[1,"Pr(>F)"]
```

```
## [1] 0.01374261
```

The F statistic for the null hypothesis $\mu_1 = \mu_2 = \mu_3$ is 5.78 against the alternative that at least one group mean differ. Under the assumption of the Gaussian linear model, F follows a Fisher distribution with 2 and 15 degrees of freedom. The P-value is thus 0.01 and we therefore reject the null hypothesis that the treatment is equal at level $\alpha = 0.05$.

The `meniscus` dat set is a balanced design, which implies that the number of observations is the same for each level of the factor. Since the entries of $\mathbf{X}^\top\mathbf{X}$ depend on the counts for each level, they will be the same here. This implies that the standard errors are also identical for each level (except for the baseline, symbolized by the intercept). Removing the latter changes the parametrization from baseline and constrast to individual group mean.

```
summary(lmod_men2 <- lm(displacement ~ 0 + method, data = meniscus))
```

```
##
## Call:
## lm(formula = displacement ~ 0 + method, data = meniscus)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -8.2667 -1.8500  0.2833  1.7708  5.3333
##
## Coefficients:
##                         Estimate Std. Error t value      Pr(>|t|)
## methodFasT-Fix            17.467      1.403  12.454 0.00000000260
## methodMeniscus Arrow      11.383      1.403   8.116 0.00000072052
```

```
## methodVertical Suture   16.950      1.403  12.085 0.00000000392
##
## Residual standard error: 3.435 on 15 degrees of freedom
## Multiple R-squared:  0.9607, Adjusted R-squared:  0.9529
## F-statistic: 122.3 on 3 and 15 DF,  p-value: 0.00000000009081
```

```
crossprod(model.matrix(lmod_men2))
```

```
##                       methodFasT-Fix methodMeniscus Arrow
## methodFasT-Fix                     6                    0
## methodMeniscus Arrow               0                    6
## methodVertical Suture              0                    0
##                       methodVertical Suture
## methodFasT-Fix                            0
## methodMeniscus Arrow                      0
## methodVertical Suture                     6
```

Note that the value of the multiple R-squared value is not the same, nor is the F-statistic. The residuals sum of square does not change because the adjustment is the same. However, without an intercept, **R** computes the decomposition $\boldsymbol{y}^\top\boldsymbol{y} = \text{RSS} + \boldsymbol{y}^\top\mathbf{H}_\mathbf{X}\boldsymbol{y}$ rather than $\text{TSS} = \boldsymbol{y}^\top\mathbf{M}_{\mathbf{1}_n}\boldsymbol{y} = \text{ESS} + \text{RSS}$. We have $\text{TSS} = \boldsymbol{y}^\top\boldsymbol{y} - n\bar{y}^2$.

We can compute the mean using the information on the total number of cases, since $\bar{y} = (\hat{\mu}_1 + \hat{\mu}_2 + \hat{\mu}_3)/3$.

5.4.2 Exercise 9.4 - Two-way analysis of variance

```
diet <- read.table("http://sma.epfl.ch/~lbelzile/math341/diet.dat", header = TRUE)
#Cast strings to factors
diet$diet.type <- as.factor(diet$diet.type)
diet$gender <- as.factor(diet$gender)

#Check mean and variance by combination of factor
#Warning: small sample sizes
facs <- list(diet$diet.type, diet$gender)
tapply(diet$weight.loss, facs, mean)
```

```
##      Female      Male
## A -3.050000 -3.650000
## B -2.607143 -4.109091
## C -5.880000 -4.233333
```

```
tapply(diet$weight.loss, facs, var)
```

```
##     Female     Male
## A 4.264231 6.431667
## B 5.239176 6.376909
## C 3.570286 7.376970
```

```
#Variance may not be homogeneous for men/women
#Fit model with interaction
M5 <- lm(weight.loss ~ gender * diet.type, data = diet)
M4 <- lm(weight.loss ~ gender + diet.type, data = diet)
```

```
M3 <- lm(weight.loss ~ diet.type, data = diet)
M2 <- lm(weight.loss ~ gender, data = diet)
M1 <- lm(weight.loss ~ 1, data = diet)
#ANOVA Tables (recall sequential decomposition) - prefered option
anova(M5)
```

```
## Analysis of Variance Table
##
## Response: weight.loss
##                  Df Sum Sq Mean Sq F value   Pr(>F)
## gender            1   0.28  0.2785  0.0518 0.820623
## diet.type         2  60.42 30.2086  5.6190 0.005456
## gender:diet.type  2  33.90 16.9520  3.1532 0.048842
## Residuals        70 376.33  5.3761
```

```
#anova(M5) uses RSS from M5 in denominator, read table from bottom to top
#RSS values
RSS <- anova(M1, M2, M3, M4, M5)[,"RSS"]
#Backward model selection (more complex to simpler)
drop1(M5, test = "F")
```

```
## Single term deletions
##
## Model:
## weight.loss ~ gender * diet.type
##                  Df Sum of Sq    RSS    AIC F value  Pr(>F)
## <none>                        376.33 133.58
## gender:diet.type  2    33.904 410.23 136.13  3.1532 0.04884
```

```
#Manual calculation
F45 <- ((RSS[4]-RSS[5])/2)/(RSS[5]/M5$df.residual)
```

For forward model selection, we select the covariate that is the most correlated with the response first. The residual sum of square of the denominator must be correct, so we could use that of the most complex model here.

```
anova(lm(weight.loss ~ gender * diet.type, data = diet))["gender", c("F value","Pr(>F)")]
```

```
##        F value Pr(>F)
## gender  0.0518 0.8206
```

```
#gender is not significant
anova(lm(weight.loss ~ diet.type * gender, data = diet))["diet.type", c("F value","Pr(>F)")]
```

```
##           F value   Pr(>F)
## diet.type  5.6292 0.005408
```

```
#diet type is significant

#Alternatively - assume that the model is CORRECT
## Clearly not the case
add1(M1, scope = c("diet.type","gender"), test = "F")
```

```
## Single term additions
##
## Model:
## weight.loss ~ 1
##           Df Sum of Sq    RSS    AIC F value   Pr(>F)
## <none>                 470.93 140.62
## diet.type  2    60.527 410.40 134.17  5.3831 0.006596
## gender     1     0.278 470.65 142.58  0.0438 0.834827
```

```
#manually computing F-test between diet.type + intercept and intercept only
isTRUE(all.equal(
  ((RSS[1]-RSS[3])/2)/(RSS[3]/(73)),
  add1(M1, scope = c("diet.type","gender"), test = "F")["diet.type", "F value"]))
```

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

```
#Note that in the above, the RSS is that of the more complex model, but not the full model
add1(M3, scope = "gender", test = "F")
```

```
## Single term additions
##
## Model:
## weight.loss ~ diet.type
##        Df Sum of Sq    RSS    AIC F value Pr(>F)
## <none>              410.40 134.17
## gender  1    0.1687 410.23 136.13  0.0296 0.8639
```

```
#No evidence against the simpler null model with only diet
#We do not test for the interaction model without main effects included!
```

We can use information criterion to estimate the goodness-of-fit of the model. The Akaike's information criterion (AIC) and Schwartz' information criterion (BIC) can be obtained by use of the functions `AIC` and `BIC`, respectively.

An alternative is Mallow's C_p, which requires an estimate of s^2. It is customary to select s^2 as the mean residual sum of squares from the most complex model, to make the comparisons fair avoid bias to due model misspecification. The most complex model will by construction give $C_p = p$ (so we cannot assess the goodness-of-fit of this model using this technique). Good models are those for which the value of C_p is close to p.

```
p <- unlist(lapply(list(M1, M2, M3, M4, M5),
                   FUN = function(mod){length(coef(mod))}))
Mallow_Cp <- RSS/RSS[5]*(nrow(diet)-p[5]) - nrow(diet) + 2 * p
Mallow_Cp
```

```
## [1] 13.596261 15.544461  6.337787  8.306409  6.000000
```

```
#Step function for model selection using AIC/BIC
#step(M5, direction = 'backward',trace = FALSE, k = log(76))

#Alternatively, compute each AIC/BIC value
aic_val <- AIC(M1, M2, M3, M4, M5)
bic_val <- BIC(M1, M2, M3, M4, M5)
which.min(aic_val$AIC)
```

```
## [1] 5
```

```
which.min(bic_val$BIC)
```

```
## [1] 3
```

```
#Store values in table
ICvals <- rbind(aic_val$AIC, bic_val$BIC)
colnames(ICvals) <- paste0("M", seq(1:5))
rownames(ICvals) <- c("AIC", "BIC")
#Create table for Markdown
knitr::kable(round(ICvals,2))
```

	M1	M2	M3	M4	M5
AIC	358.30	360.26	351.85	353.81	351.26
BIC	362.96	367.25	361.17	365.47	367.57

```
#Create table to export in LaTeX
tab <- xtable::xtable(round(ICvals,1), caption = "Two-way ANOVA, diet data")
#print(tab, booktabs = TRUE)
```

Chapter 6

Hypothesis testing

This is a refresher on the notions related to hypothesis testing.

Trial analogy: suppose you are a member of the jury for a trial where the potential culprit stands accused of murder. If you declare him guilty, the sentence will likely be death penalty. The null hypothesis is innocence: the accusee is innocent until proven guilty. You will deliver a verdict of culpability only if the evidences are overwhelming against the accusee (you do not want to send an innocent to the death row and make a wrongful conviction).

In this setting, the verdict at the end of the trial will reflect "this innocent until proven guilty" mindset: you can usually only conclude that there are not enough proofs to sentence the accusee of murder, not that the person is innocent. This is why we "fail to reject the null hypothesis", since you gave the accusee the benefit of the doubt in the first place and examined the evidence in this optic.

Test statistics are realizations of random variables, so the **size** of a test is the probability of falsely rejecting the null hypothesis,

$$\alpha = \Pr(\text{reject } \mathrm{H}_0 \mid \mathrm{H}_0 \text{ is true}).$$

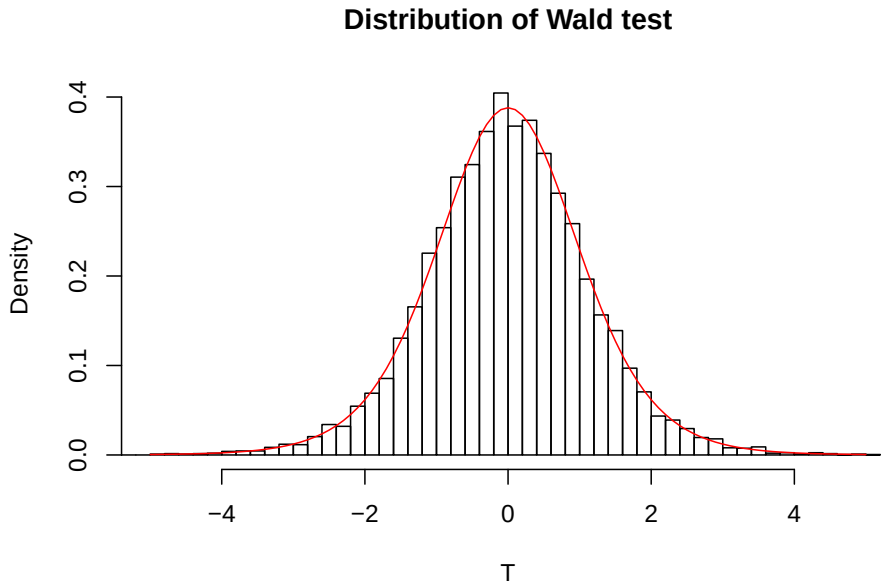
This is fixed beforehand: if we take $\alpha = 0.05$, there will be 95% of the cases we will correctly release an innocent and 5% of the cases where we will convict him undully (due to circumstancial factors, for example).

We illustrate various concepts with the simple model

$$Y_i = \beta_0 + \varepsilon_i, \qquad \varepsilon_i \overset{\text{iid}}{\sim} \mathcal{N}(0, \sigma^2) \qquad (i = 1, \ldots, n)$$

The Wald test statistic for the null hypothesis $\mathrm{H}_0 : \beta_0 = 0$ against the alternative $\mathrm{H}_a : \beta_0 \neq 0$ is $t = \hat{\beta}_0 / \mathrm{se}(\hat{\beta}_0) \sim \mathcal{T}(n-p)$. We can compare the Student distribution with the empirical distribution of t-test obtained by simulating a large number of test statistics from the model; these should match.

```
test <- rep(0, 10000L)
n <- 10L
#Simulate Normal samples, compute t-test stat
for(i in 1:length(test)){
  y <- rnorm(n)
  test[i] <- mean(y) / (sd(y)/sqrt(n))
}
#Create histogram, superimpose density of T(n-1)
hist(test, breaks = 50, probability = TRUE,
     main = "Distribution of Wald test", xlab = "T",
     xlim = c(-5, 5))
curve(dt(x, df = n - 1), col = 2, add = TRUE,
      from = -5, to = 5)
```



If the value $|t|$ is very large, then there are evidences that $\beta_0 \neq 0$. In this case, the probability of observing something larger than $|t|$ under $T \sim \mathcal{T}(n-p)$ is $P = 1-\Pr(-t < T < t) = 1-2\Pr(|T| < t)$, by virtue of the symmetry of the Student distribution. This probability P is called P-**value**, the probability of observing something as extreme under the null distribution.

The **power** of a test is

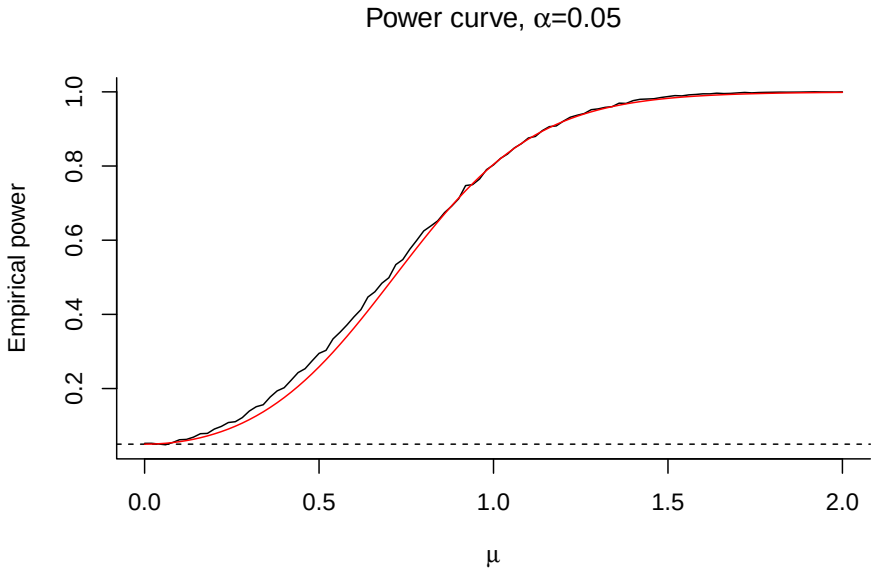
$$\text{power} = \Pr(\text{reject } H_0 \mid H_a \text{ is true}).$$

Consider the alternative $H_a : \beta = \mu \neq 0$. For the t-test, the power is a function of μ, σ^2 and n. Intuitively, the further μ is from zero, the larger the chance of correctly detecting that $\mu \neq 0$. Similarly, the more precise our mean estimate is (when σ^2 is small), the more we have. Lastly, evidence accumulates with the sample size - here through the degrees of freedom parameter.

Even if we don't know the distribution of the test statistic under the alternative, we can simulate the power curve as a function of μ, σ and n.

```
n <- 10L
nrep <- 10000L
mu <- seq(0, 2, length = 101)
store <- matrix(0, nrow = length(mu), ncol = nrep)
for(i in 1:length(mu)){
  for(j in 1:nrep){
    # Simulate y = mu + eps, eps ~ N(0,1)
    y <- rnorm(n, mean = mu[i])
    tstat <- mean(y) / (sd(y)/sqrt(n))
    # Compute P-value, i.e., probability Pr(|T| > t)
    pval <- 2*(1 - pt(abs(tstat), df = n - 1))
    store[i, j] <- pval
  }
}
# Compute the proportion of time where p-value is below zero.
plot(mu, rowSums(store < 0.05)/nrep,
     type = "l", xlab = expression(mu),
     ylab = "Empirical power", bty = "l",
     main = expression(paste("Power curve, ", alpha, "=0.05")))
#size of test
```

```
abline(h = 0.05, lty = 2)
#theoretical power - derivation follows
tquant <- qt(0.975, df = n - 1)
pow <- 1 - (pt(tquant + mu*sqrt(n), df = n - 1) -
                pt(-tquant + mu*sqrt(n), df = n - 1))
lines(mu, pow, col = 2)
```



Under H_0, our test statistic $T = \hat{\beta}_0/\text{se}(\hat{\beta}_0)$ followed a $\mathcal{T}(n-1)$ distribution and the cutoff value was $t_{1-\alpha/2}$, so that under H_0, $\Pr(|T| > t_{1-\alpha/2}) = \alpha$.

We can compute the power exactly as a function of μ in this example: it is

$$\begin{aligned}\beta(\mu) &= 1 - \Pr\left(t_{1-\alpha/2} \le T \le t_{1-\alpha/2}; H_a\right) \\ &= 1 - \Pr\left(t_{1-\alpha/2} \le \frac{\hat{\beta}_0 - \mu + \mu}{\text{se}(\hat{\beta}_0)} \le t_{1-\alpha/2}; H_a\right) \\ &= 1 - \Pr\left(t_{1-\alpha/2} + \frac{\mu}{\text{se}(\hat{\beta}_0)} \le \frac{\hat{\beta}_0 - \mu}{\text{se}(\hat{\beta}_0)} \le t_{1-\alpha/2} + \frac{\mu}{\text{se}(\hat{\beta}_0)}; H_a\right).\end{aligned}$$

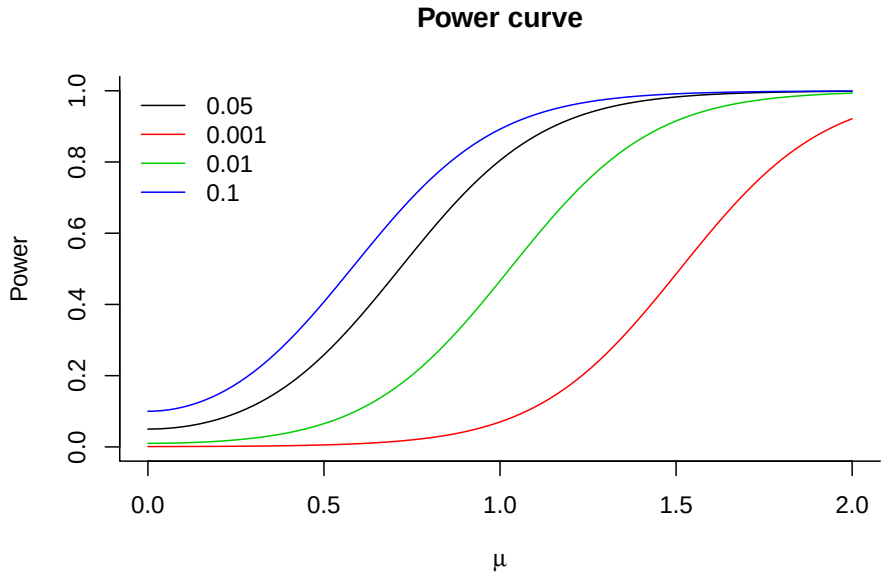
since now $T^* = (\hat{\beta}_0 - \mu)/\text{se}(\hat{\beta}_0) \sim \mathcal{T}(n-1)$ under H_a. If we superimpose this curve as a function of μ, we see it matches the empirical power.

The power curve at $\mu = 0$ is 0.05, since the size of the test is $\alpha = 0.05$ in this experiment. If we increase the size of the test, then power increases:

The probability of Type I error (falsely rejecting the null) is the size of the test, so increases with α. The lower the α, the higher the probability of Type 2 errors (not rejecting the null when the alternative is true) and the lower the power.

```
plot(mu, pow,
     type = "l", xlab = expression(mu),
     ylab = "Power",
     ylim = c(0, 1), bty = "l",
     main = "Power curve")
 alphav <- c(0.05, 0.001, 0.01, 0.1)
```

```
for(alpha_ind in 2:4){
  alpha <- alphav[alpha_ind]
  tquant <- qt(1-alpha/2, df = n - 1)
  pow <- 1 - (pt(tquant + mu*sqrt(n), df = n - 1) -
                  pt(-tquant + mu*sqrt(n), df = n - 1))
lines(mu, pow, col = alpha_ind)
}
legend(x = "topleft", legend = alphav ,col = 1:4, bty = "n", lty = rep(1,4))
```



Chapter 7

Model selection

Why perform model selection? Practitionners tend to prefer the use of a single model to model aggregation or averaging, since having a single model is easier to interpret. One guiding principle for choosing a single model is parsimony and interpretability: we want a model that is not overly complex and that does not overfit the data. This is most important if the goal of the analysis is prediction. Whenever possible, we will make comparisons between nested models.

My (personal) general strategy for model selection, given the tools covered in MATH 341, is the following:

1. Start with a complex model (all additive terms, say). Look at the individual Wald tests for the marginal significance of the coefficients.
2. Try to simplify the full model using an F test, potentially dropping multiple terms at once. This preserves power (avoids the potential bias in the estimate of RSS in the denominator; cf. 5.1.3 and reduces the multiple testing problem that inflates Type I errors (reject the null more than α% of the time when you shouldn't).
3. Repeat; you can use `drop1` to further reduce the model.
4. Try adding interactions between variables, if any.
5. Compare the nested models using an information criterion such as AIC or BIC.
6. Select the best forward selection / backward elimination models and some additional ones (that have a nice interpretation, have low information criterion values, etc.) Compare them in terms of prediction and goodness-of-fit.

Once you have a final model, you can interpret coefficients. Model selection **invalides** classical inference, so report coefficients and standard errors as is without overinterpreting the P-values from `summary`.

The best tool to assess the predictive power your model is the use of cross-validation. If there is no temporal structure, you can use e.g. five-fold cross validation to find the best fitting model.

7.1 Example: Price of diamonds

This very large dataset contains the price in USD (rounded to nearest dollar) of $n = 53940$ diamonds. The explanatory variables include three ordinal factors: the quality of the `cut`, `color` and `clarity`. These are ranked from worst to best outcome. Five other variables contain the mensurements of the dimension of the diamond, rounded to the 0.01 mm. They are length `x`, width `y`, depth `z`, total depth percentage `depth` where `depth`$= 2 \times z/(x+y)$, and `table`, a measure of the width of the top of the diamond. The last variable is the weight of the diamond, `carat`, rounded to the nearest 0.01.

```
#install.packages("ggplot2")
library(ggplot2); library(car)
```

```
data(diamonds, package = "ggplot2")
help(diamonds)
#Subsample because the dataset is very large
set.seed(1234) #Fix RNG seed so as to make output reproducible
di <- diamonds[sample.int(size = 500, replace = FALSE, n = nrow(diamonds)), ]
attach(di)
```

7.1.1 Exploratory data analysis

We can look at some graphs of the data, including pair plots and some summary statistics. These are useful to spot outliers.

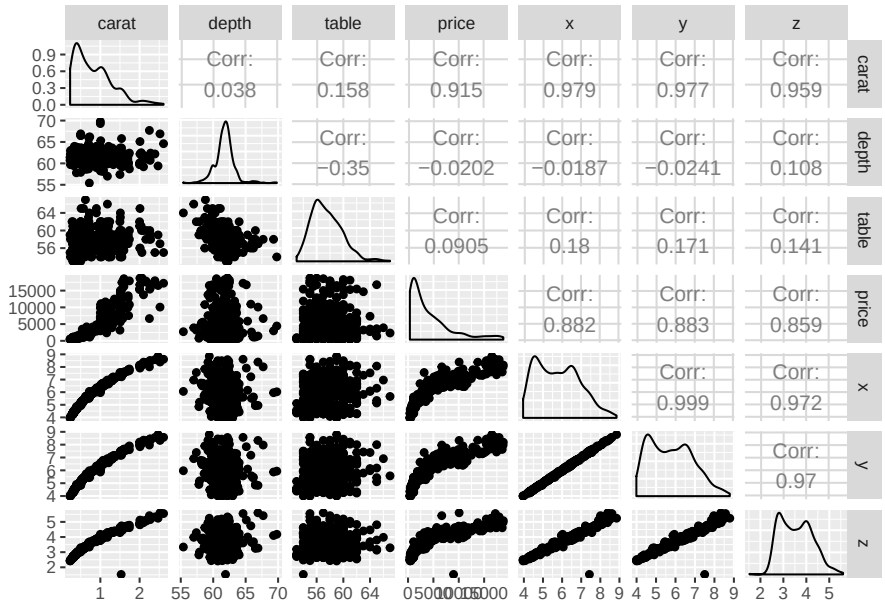
```
str(di)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    500 obs. of  10 variables:
##  $ carat  : num  0.91 0.43 0.32 0.33 0.7 0.33 0.71 1.3 0.43 2.06 ...
##  $ cut    : Ord.factor w/ 5 levels "Fair"<"Good"<..: 5 4 5 5 2 5 4 2 4 5 ...
##  $ color  : Ord.factor w/ 7 levels "D"<"E"<"F"<"G"<..: 4 1 1 4 5 4 2 7 2 6 ...
##  $ clarity: Ord.factor w/ 8 levels "I1"<"SI2"<"SI1"<..: 2 3 4 2 3 7 4 4 2 4 ...
##  $ depth  : num  61.6 60.1 61.5 61.7 64.2 61.8 62.3 63.6 60.8 62.2 ...
##  $ table  : num  56 58 55 55 58 55 58 59 58 55 ...
##  $ price  : int  3985 830 808 463 1771 868 2823 5269 919 18779 ...
##  $ x      : num  6.24 4.89 4.43 4.46 5.59 4.42 5.71 6.9 4.88 8.15 ...
##  $ y      : num  6.22 4.93 4.45 4.48 5.62 4.45 5.66 6.87 4.86 8.19 ...
##  $ z      : num  3.84 2.95 2.73 2.76 3.6 2.74 3.54 4.38 2.96 5.08 ...
```

```
apply(di, 2, range)
```

```
##      carat  cut         color clarity depth  table  price   x      y
## [1,] "0.23" "Fair"      "D"   "I1"    "55.4" "53.0" "  364" "3.96" "3.99"
## [2,] "2.61" "Very Good" "J"   "VVS2"  "69.8" "67.0" "18779" "8.85" "8.79"
##      z
## [1,] "1.53"
## [2,] "5.60"
```

```
print(GGally::ggpairs(di[,-(2:4)], progress = FALSE))
```




```
table(cut)
```

```
## cut
##      Fair      Good Very Good   Premium     Ideal
##        16        46       100       123       215
```

```
table(clarity)
```

```
## clarity
##    I1   SI2   SI1   VS2   VS1 VVS2 VVS1    IF
##     7    89   110   118    76    44    36    20
```

```
table(color)
```

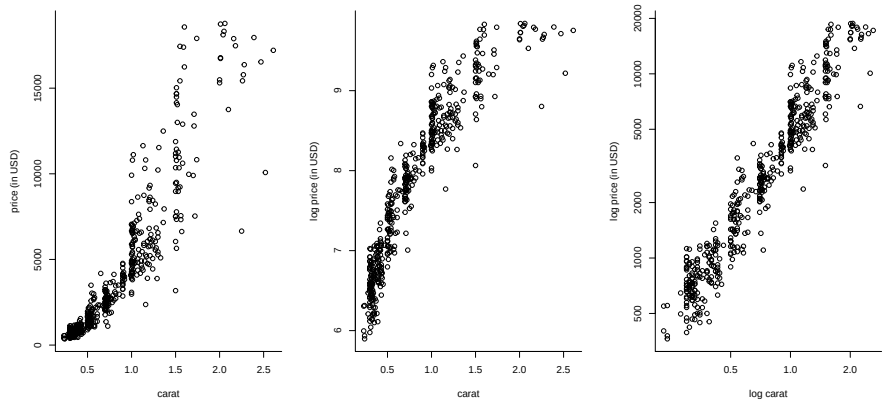
```
## color
##    D    E    F    G    H    I    J
##   57  100   77   89   95   52   30
```

```
graphics.off()
plot(density(carat, bw = 0.02), main = "Density estimate of carat")
```

The most important variable is likely to be weight or width (which are strongly correlated). An explanatory data analysis reveals the relationship between `carat` and `price` to be non-linear. A logarithmic transformation of both the `price` and `carat` alleviates this and reveals the discretization of the measurements (most of the diamonds have a reported weight of 1 or 2 carats). The linear correlation between the variables `x`, `y` and `z` is close to unity (due to the regular cut of diamonds). This will potentially lead to collinearity, so the variables may not be jointly significative. There is (depending on the subset) a clearly visible outlier in `z` hat should be removed. There is no evidence of interactions between the categorical variables and the rest (not shown). Lastly, `depth` and `table` are apparently not linearly correlated with `price`.

```
par(mfrow = c(1, 3), bty = "l")
plot(x = carat, y = price, ylab = "price (in USD)", xlab = "carat")
```

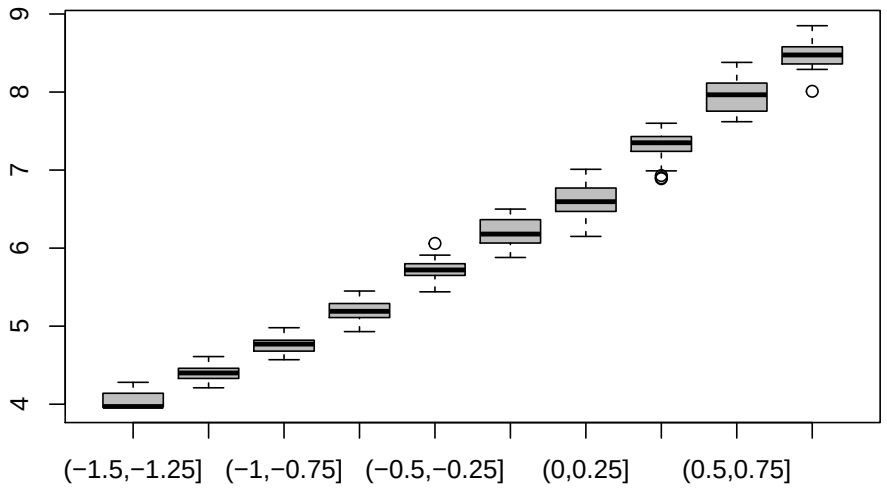
```
plot(carat, log(price), ylab = "log price (in USD)", xlab = "carat")
plot(carat, price, log="xy", ylab = "log price (in USD)", xlab = "log carat")
```



A careful explanatory data analysis with the full model reveals that, despite the fact `depth` is a transformed variable supposedly created from `x`, `y` and `z`, there are some outliers (`summary(lm(depth ~ -1 + I(z/(x+y)), data = diamonds))`). The model is likely to predict poorly 1 carat and 2 carats diamonds, for which there is a lot of heterogeneity.

Sometimes, it helps to regroup the regressors to better identify patterns. This is most useful in situations where there is a lot of noise in the response (not the case here).

```
lcarat_cut <- cut(log(carat), breaks = seq(-1.5, 1, by = 0.25))
boxplot(x ~ lcarat_cut, col = 'grey')
```



7.1.2 Model selection

We will start with a model with all regressors but `y` and `z` (which we eliminate on grounds of multicollinearity).

```
#Small model
redu_mod <- lm(log(price) ~ log(carat))
#Unordered factors (so they are interpretable)
#Ordered factors use an orthogonal decomposition
```

```
cut <- factor(cut, ordered = FALSE)
color <- factor(color, ordered = FALSE)
clarity <- factor(clarity, ordered = FALSE)
#Full additive model
full_mod <- lm(log(price) ~ log(carat) + cut + color + clarity + depth + table + x)
summ_full <- summary(full_mod)
knitr::kable(coef(summ_full), digits = 3)
```

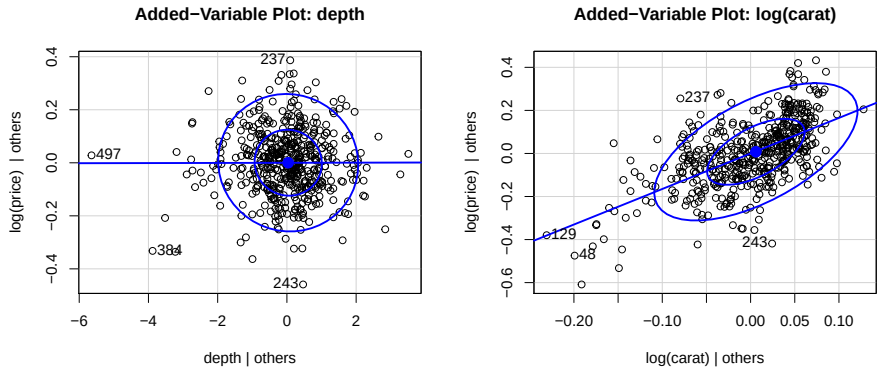
	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	7.159	0.698	10.255	0.000
log(carat)	1.651	0.101	16.283	0.000
cutGood	0.118	0.040	2.964	0.003
cutVery Good	0.114	0.040	2.878	0.004
cutPremium	0.143	0.039	3.641	0.000
cutIdeal	0.142	0.042	3.404	0.001
colorE	-0.056	0.021	-2.604	0.009
colorF	-0.091	0.023	-3.967	0.000
colorG	-0.161	0.022	-7.233	0.000
colorH	-0.270	0.022	-12.233	0.000
colorI	-0.400	0.026	-15.568	0.000
colorJ	-0.547	0.030	-18.422	0.000
claritySI2	0.659	0.053	12.388	0.000
claritySI1	0.807	0.053	15.180	0.000
clarityVS2	0.952	0.053	17.940	0.000
clarityVS1	1.035	0.054	19.192	0.000
clarityVVS2	1.144	0.056	20.457	0.000
clarityVVS1	1.224	0.057	21.461	0.000
clarityIF	1.341	0.060	22.164	0.000
depth	0.000	0.005	0.049	0.961
table	-0.007	0.004	-1.634	0.103
x	0.132	0.053	2.522	0.012

```
RSS_full <- summ_full$sigma^2 * summ_full$df[2]
#RSS_full <- crossprod(resid(full_mod))[1]
```

The model fit is excellent. Unsurprisingly, all of `x`, `y` and `z` are not marginally significant if they are all included at once (output omitted), but `x` is if it is the only one included.

Recall that the `t value` column gives the Wald statistic $t = \hat{\beta}_i / \text{se}(\hat{\beta}_i)$, for the null hypothesis $\beta_i = 0$ against the two-sided alternative $\beta_i \neq 0$. Under $\mathcal{H}_0$, $t \sim \mathcal{T}(n-p)$ and the P-value is $2 \times (1 - \texttt{pt}(t, n-p))$. It appears that we could get rid of `depth` and `table`, which contribute little overall. This is confirmed graphically using added-variable plots, which plots $\mathbf{M}_{\mathbf{X}_{-j}} \boldsymbol{y}$ against $\mathbf{M}_{\mathbf{X}_{-j}} \mathbf{x}_j$. This is the residual effect of $\mathbf{x}_j$ after taking into account the effect of the other variables $\mathbf{X}_{-j}$ on what remains of the response. If the variable was important, there would be a strong correlation in the variables and the slope would be non-zero. The last plots illustrates what you could see if there was residual structure (strong positive or negative correlation) or lack thereof.

```
par(mfrow = c(1, 2))
car::avPlot(full_mod, variable = "depth", ellipse = TRUE)
#slope close to zero indicates lack of relationship
car::avPlot(full_mod, variable = "log(carat)", ellipse = TRUE)
```



Let us look at model simplifications. We can obtain the F statistic for the null hypothesis $\mathcal{H}_0 : \beta_{\texttt{depth}} = \beta_{\texttt{table}} = 0$ against the alternative $\mathcal{H}_a : \{(\beta_{\texttt{depth}}, \beta_{\texttt{table}}) \in \mathbb{R}^2\}$ by running the `anova` command:

```
anova(lm(log(price) ~ log(carat) + cut + color + clarity + x), full_mod)
```

```
## Analysis of Variance Table
##
## Model 1: log(price) ~ log(carat) + cut + color + clarity + x
## Model 2: log(price) ~ log(carat) + cut + color + clarity + depth + table +
##     x
##   Res.Df    RSS Df Sum of Sq      F Pr(>F)
## 1    480 7.8077
## 2    478 7.7503  2  0.057353 1.7686 0.1717
```

The test statistic is of the form

$$F = \frac{(\text{RSS}_a - \text{RSS}_0)/2}{\text{RSS}_0/478} \sim \mathcal{F}(2, 478)$$

and here $F = 1.769$; we fail to reject the null (P-value of 0.172). We have no evidence against the adequacy of the simpler model.

Since there is little difference between the reduced model RSS and that of the full additive model, we may employ either in subsequent ANOVA tests. Let us try to drop one of the remaining variables.

```
drop1(lm(log(price) ~ log(carat) + cut + color + clarity + x), test = "F")
```

```
## Single term deletions
##
## Model:
## log(price) ~ log(carat) + cut + color + clarity + x
##            Df Sum of Sq     RSS     AIC  F value    Pr(>F)
## <none>                   7.8077 -2039.8
## log(carat)  1    5.7660 13.5737 -1765.2 354.4787 < 2.2e-16
## cut         4    0.4411  8.2488 -2020.3   6.7788 0.0000259
## color       6   10.1359 17.9436 -1635.7 103.8552 < 2.2e-16
## clarity     7   18.5675 26.3752 -1445.1 163.0697 < 2.2e-16
## x           1    0.0961  7.9038 -2035.6   5.9062   0.01545
```

You could write the test statistics as before (with the difference in the degrees of freedom equal to the number of factor levels minus one if the variable is categorical). All the terms are statistically significant and we reject the null hypothesis of any of the tests at level $\alpha = 5\%$. This step would mark the end of the backward elimination procedure. Note that you can (and may wish to) use the RSS from the full model `full_mod` in the denominator of your F-tests to avoid biasing your results if retaining the null leads to a sharp decrease.

If your test statistic is small, you cannot conclude anything. This may be because the null hypothesis that the simpler model is adequate is true. It can also be due to a lack of power (you should reject, but there are not enough evidences against the null). If you proceed with the RSS from the null model, your test statistic will then be biased downward; recall section 5.1.3.

We could equally well have started with forward selection. All the variables lead to a decrease in RSS that is significant at level $\alpha = 5\%$. We pick the most significant one and proceed.

```
add_step1 <- add1(redu_mod, scope = formula(full_mod), scale = RSS_full, test = "F")
form <- deparse(formula(redu_mod))
while(length(which(add_step1[,"Pr(>F)"] < 0.05)) > 0){
  new_var <- rownames(add_step1)[which.max(add_step1[,"F value"])]
  form <- paste(form, new_var, sep = " + ")
  add_step1 <- add1(update(redu_mod, formula = form),
                    scope = formula(full_mod), scale = RSS_full, test = "F")
}
```

The more we test using the ANOVA command, the more size distortion due to multiple testing (the type I error is inflated). A Bonferroni correction could alleviate this. Note that forward selection typically uses a biased estimate of the residual sum of square.

The variable that is the most correlated with `log(price)` is `clarity` and leads to a significant increase, so we would go for the bigger model since there is strong evidence that the model fit is better.

Both forward selection and backward elimination yielded the same model, with the three categorical variables and length. At this stage, we should try and include interactions.

```
add1(lm(formula = form, data = di),
        scope =  as.formula(paste(form, "+ log(carat):color + log(carat):clarity +
                                  log(carat):cut + color:cut + color:clarity + cut:clarity", collapse =
     test = "F")
```

```
## Single term additions
##
## Model:
## log(price) ~ log(carat) + clarity + color + x + table + cut
##                     Df Sum of Sq    RSS     AIC F value     Pr(>F)
## <none>                           7.7504 -2041.4
## log(carat):color     6   0.12893 7.6215 -2037.8  1.3336   0.240428
## log(carat):clarity   7   0.17840 7.5720 -2039.1  1.5886   0.136476
## log(carat):cut       4   0.39334 7.3570 -2059.5  6.3489 0.00005540
## color:cut           23   0.44771 7.3027 -2025.2  1.2155   0.225315
## clarity:color       37   1.22732 6.5231 -2053.6  2.2476 0.00006789
## clarity:cut         21   0.65138 7.0990 -2043.3  2.0012   0.005557
```

The interaction between `log(carat)` and `cut` is significant at the 5% level, idem for `clarity:color` and `clarity:cut`. Keep in mind that adding interactions leads to a large increase in the number of parameters; `clarity:color` would lead to an additional 37 parameters!

7.1.3 Information criterion

We have covered (old school) partial F-tests in the ANOVA section. Other widely (mis)used goodness-of-fit diagnostics are AIC and BIC. These information criterion are goodness-of-fit measures coupled with model complexity penalty. They are (under many hypothesis) estimates of the Kullback–Leibler divergence.

Akaike's An Information Criterion (AIC) is AIC $= -2\ell(\hat{\boldsymbol{\theta}}) + 2p$, while Schwartz's information criterion is BIC $= -2\ell(\hat{\boldsymbol{\theta}}) + p\log(n)$. The latter is more stringent and penalizes more heavily the complex models as more data becomes available.

Some general remarks if you use information criteria

1. AIC and BIC *must be* computed using maximum likelihood estimators. In a linear model, this means that the estimator of the variance is $\hat{\sigma}^2 = \text{RSS}/n$ and not s^2. Similarly, the ordinary least square estimator is equivalent to the MLE for $\hat{\boldsymbol{\beta}}$ if $\text{Va}(\boldsymbol{\varepsilon}) = \sigma^2\mathbf{I}_n$. In **R**, you can use `BIC` and `AIC` commands on models obtained from `lm` to get those values.
2. The models should include the *same data* to be comparable.
3. If you are comparing different distributions, you need to include all the constants to make AIC values comparable.
4. The models need not be nested, but there are consequences if they are not. The AIC $= \text{O}_p(n)$ (so information accumulates over time at a linear rate), but its variability is $\text{O}_p(n^{1/2})$ — this difference is only $\text{O}_p(1)$ when models are nested.
5. If the *TRUE* model exists and one of the models under consideration, it will be selected as $n \to \infty$ with probability one by BIC and with high probability by AIC.

The function `step` allows you to do forward or backward model selection using one of the information criterion. I **do not** recommend this, as there is no theoretical ground for the method. Comparisons based on information criteria are mostly meaningful if the models are nested. You may wish to use BIC rather than AIC because the latter leads to more parsimonious (and consistent model selection in the large sample limit). It may be a good starting point for your model search.

```
BIC_search <- formula(step(full_mod, trace = 0, k = log(nrow(di)))) #BIC
#Search within space of main effect-only models
BIC_mod <- lm(BIC_search, data = di)
final_mod <- lm(log(price) ~ log(carat) + cut + color + clarity +
                  x + table + log(carat):cut, data = di)
#summary(final_mod)
```

A further simplification would consist in merging the factors levels, typically into low quality and high quality. This may not a good idea, because it will disproportionally affect the prediction of large diamonds worth a lot, and will negatively impact the predictive accuracy. To merge factors, use e.g.

```
newcut <- cut
levels(newcut) <- list("Fair-Good" = c("Fair", "Good"), "High" = c("Very Good", "Premium", "Ideal"))
```

To export your table to LaTeX, I recommend you use dedicated packages such as `texreg`, `stargazer` or `xtable`. You can easily export, using e.g. the command `texreg::texreg(final_mod, stars = 0, digits = 2, single.row = TRUE, booktabs = TRUE)`, to get what you want. The level of customization is important (so you could rename the columns). Please make sure the font size is adequate and you use the right amount of digits.

7.1.4 Cross-validation

Let us now compare the predictive performance of the model using cross-validation. The idea underlying cross-validation is simple: split the data, use a fraction (called training set) for model fit and the remaining observations (termed validation set) to check predictions.

The predicted residual error sum of squares (PRESS), denoted CV in the course, is the result of leave-one-out cross validation. The ith observation is predicted using the $n-1$ other observations for every $i = 1, \ldots, n$. That is, we do not use the observation both for estimation and prediction and thus the predicted residual error is a more accurate measure of prediction error. We can return the PRESS statistic using the residuals from **R**.

```
#Leave-one-out cross validation
PRESS <- function(model){crossprod(rstandard(model, type = "pred"))[1,1]}
round(c("reduced model" = PRESS(redu_mod), #underfit?
        "final model " = PRESS(final_mod),
        "full model" = PRESS(full_mod)),
  digits = 2) #overfitting
```

```
## reduced model  final model     full model
##         38.58          8.29          8.61
```

The cross-validated error estimate shows that we do significantly better with the final model than using simply the model with `log(carat)` and that the addition of `x` does not increase predictive accuracy. The full model has a larger prediction error, an indication that we may be overfitting.

Rather than use only one observation for validation and $n-1$ for training, we can split more evenly: K-fold cross validation uses $n - n/K$ observations for fitting and n/K for validation, providing a more realistic depiction of prediction. Unfortunately, the number of possible subsets of size $\lfloor n/K \rfloor$ is very large and so one typically split the data into classes of equal size at random. The following function, which performs K-fold cross validation, can be used in your project. Since the result is random, it may be necessary to average over many replicates of the K-fold statistic provided that the calculation is not too computationally demanding. For large n, this has less impact.

The smaller the prediction error, the better the model.

```
K <- 10
#Manually perform cross fold validation
KfoldCV <- function(fitted.mod, K, ...){
  data <- model.matrix(fitted.mod) #design matrix
  y <- fitted.mod$model[,1] #response
  n <- nrow(data)
  #Shuffle the indices
  inds <- sample.int(n = n, size = n, replace = FALSE)
  #Split into K groups of ~ equal size (from https://stackoverflow.com/a/16275428)
  form_group <- function(x, n){ split(x, cut(seq_along(x), n, labels = FALSE)) }
  groups <- form_group(inds, K)
  #Obtain prediction from K-folds
  preds <- rep(NA, n)
  for(j in 1:K){
   preds[groups[[j]]] <- data[groups[[j]],] %*%
                           lm(y[-groups[[j]]] ~ -1 + data[-groups[[j]],])$coef
  }
  #Compute prediction error
  crossprod(preds - y)[1,1]
}
## Because splitting is random, get different answer
round(c("reduced model" = median(replicate(KfoldCV(fitted.mod = redu_mod, K = K), n = 100)),
        "final model " = median(replicate(KfoldCV(fitted.mod = final_mod, K = K), n = 100)),
        "additive forward" = median(replicate(KfoldCV(fitted.mod = BIC_mod, K = K), n = 100))),
        digits = 2)
```

```
##    reduced model     final model  additive forward
##            38.60            8.34            8.74
```

The conclusions are the same for 10-fold cross validation as for leave-one-out cross validation, conforting our model choice. In general, we prefer the former.

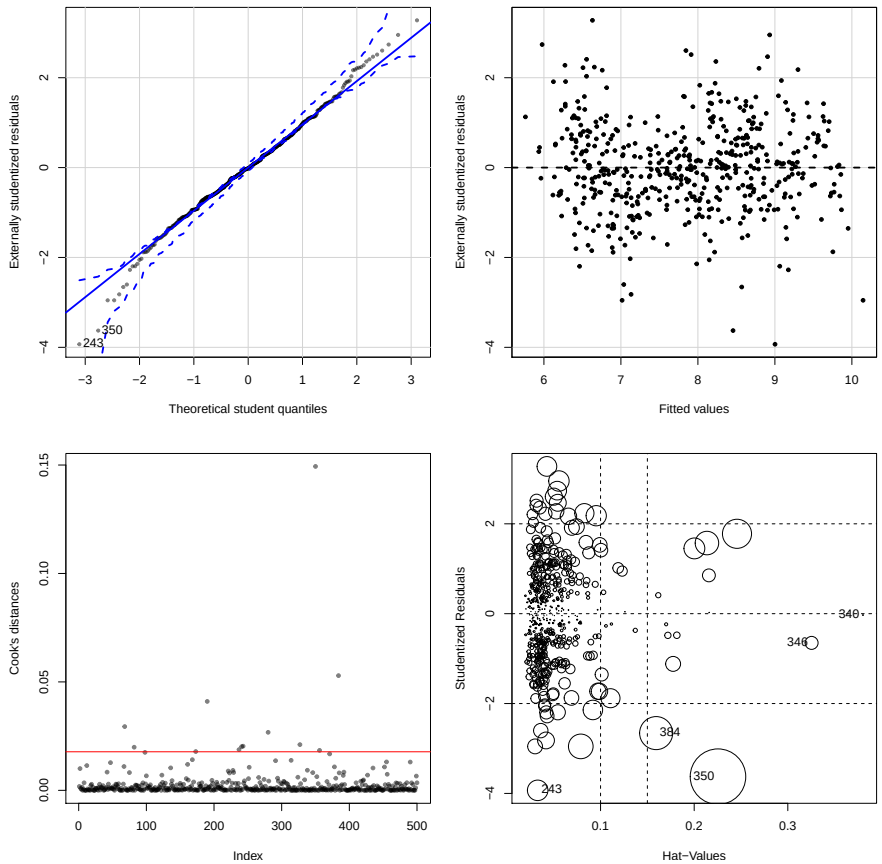
7.1.5 Presentation of results

Having selected a model (say `final_mod`), you should now present a table with the coefficients and standard errors, some goodness-of-fit measures (R^2_c, AIC/BIC, CV, K-fold cross-validation error). Explain your model (interpret the parameters), look at diagnostic plots and answer the questions.

```
par(mfrow = c(2, 2), mar = c(5, 5, 1.5, 0.5))
bl <- scales::alpha("black", 0.5) #semi-transparent black
n <- nrow(di)
#plot(final_mod)

#Student Q-Q plot
qqPlot(final_mod, simulate = 1999, envelope = TRUE,
       ylab = "Externally studentized residuals",
       xlab = "Theoretical student quantiles",
       pch = 20, col = bl)
```

```
## [1] 243 350
```

```
#Residuals vs fitted values
residualPlot(final_mod, type = "rstudent", quadratic = FALSE,
             pch = 20, ylab = "Externally studentized residuals")
#Cook distance
plot(cooks.distance(final_mod), col = bl, pch = 20, ylab = "Cook's distances")
abline(h = 8/(n-2*length(coef(final_mod))), col = 2)
influencePlot(final_mod)
```



```
##         StudRes        Hat         CookD
## 243 -3.93155877 0.03281988 0.02036095303
## 340 -0.03069107 0.38041277 0.00002318201
## 346 -0.64996148 0.32541064 0.00816123471
## 350 -3.62756784 0.22541782 0.14936005917
## 384 -2.65776683 0.15938312 0.05289674514
```

Many points have high leverage and large Cook's values. This means the slope could in principle largely be driven by those points We get a very large residual (observation 350, which is a very small diamond of high quality sold for almost double the value of a comparable one). A more careful analysis would be necessary to see the impact of these points on the coefficients and whether they matter (or not). For example, we could refit the model using the command `lm(final_mod, subset = -350)`.

Diagnostics of heteroscedasticity are mostly useful when you have suspicions that different subgroups could have different variances (if you include factor variables) or if the data are time series, in which case you may wish to look at plots of the correlogram (`acf(resid(final_mod))` and partial correlogram `pacf(resid(final_mod))`). These are only useful for time ordered observations, i.e. when the errors at time t depend on previous time periods $\{t-1,\ldots\}$. The impact of serial dependence of the residuals is that the standard errors from OLS are too small and need to be inflated.

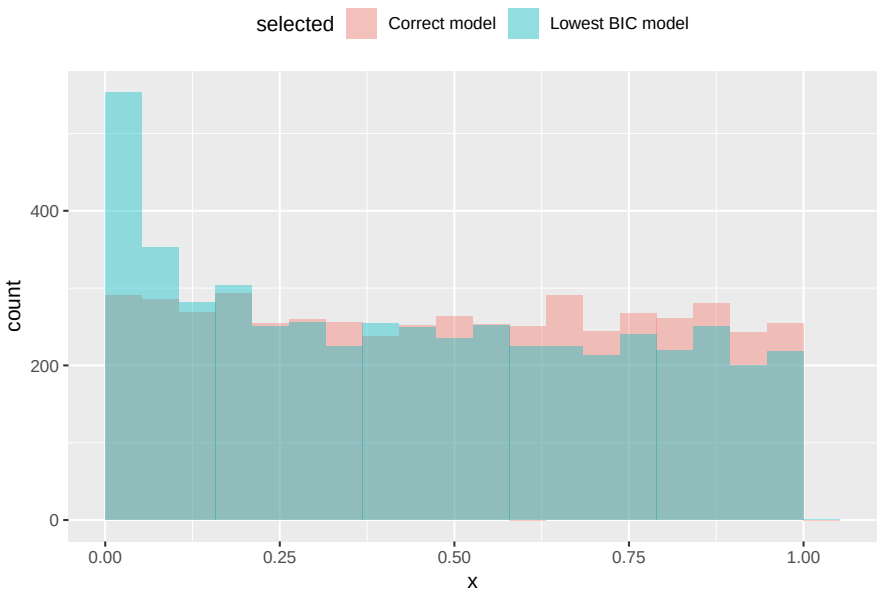
7.2 Model selection invalidates P-values

Variable selection invalidates the usual Student and Fisher tests. To see this, we can generate a dataset $\boldsymbol{y} = \boldsymbol{\varepsilon}$ and create an $n \times p$ design matrix $\mathbf{X}$ with random uncorrelated elements. We can assess by means of simulation the size distortion (the difference between the nominal level and the type I error) of a t test for the hypothesis $\beta_j = 0$ after performing forward selection.

We illustrate the computations using two models: one in which the variables in the design matrix, $\mathbf{X}$, are strongly correlated and another with independent input using BIC and Mallow's C_p as selection criterion. The size distortion is comparable.

```
## Uncorrelated variables, model selection using BIC
```

```
##                   size
## model                0.1   0.05   0.01
##   Lowest BIC model 0.1738 0.1076 0.0324
##   Correct model    0.1104 0.0562 0.0106
```




```
## Correlated variables, model selection using BIC
```

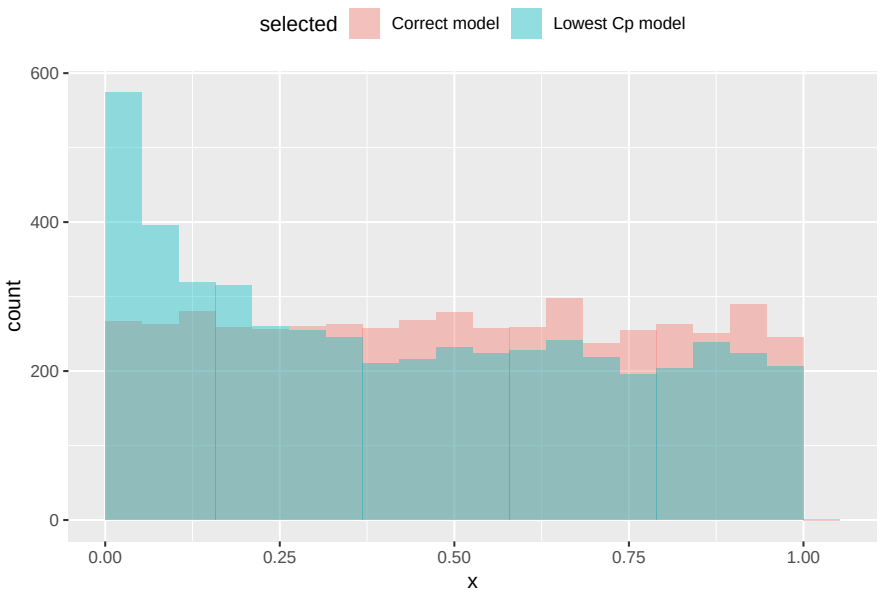
```
##                   size
## model                0.1   0.05   0.01
##   Lowest BIC model 0.1846 0.1150 0.0344
##   Correct model    0.1018 0.0504 0.0110
```

```
## Uncorrelated variables, model selection using Mallow's Cp
```

```
##                  size
## model               0.1   0.05   0.01
##   Lowest Cp model 0.1844 0.1096 0.0318
##   Correct model   0.0978 0.0472 0.0088
```

```
## Correlated variables, model selection using Mallow's Cp
```

```
##                  size
## model               0.1   0.05   0.01
##   Lowest Cp model 0.1864 0.1114 0.0352
##   Correct model   0.1014 0.0514 0.0084
```



Adding superfluous variables does not induce bias, but inflates the standard errors (leading to larger P-values). We observe on the contrary here smaller P-values. Recall that the P-values should be uniformly distributed under the null hypothesis (and they seemingly are when the true null model is fitted).

Chapter 8

Penalized regression methods

There are two reasons one may go down the route of penalized regression models: one is to trade-off bias for reduced variance in the hope that doing so will improve the predictive accuracy of the model. The other is in cases where the number of covariates p is large and the matrix $\mathbf{X}$ is close to being non-invertible.

Throughout, we will work with centered and standardized inputs. One reason for doing so is to make our inference invariant to affine transformations, much like for R_c^2. We can consider a design matrix $[\mathbf{1}_n\ \mathbf{X}]$ with rescaled columns so that $\text{Var}(\mathbf{X}_j) = 1$ for $j = 1, \ldots, p$ and take $\mathbf{Z} \equiv \mathbf{X} - \mathbf{1}_n\bar{\mathbf{X}}$, where $\bar{\mathbf{X}}$ is the row vector of column means of $\mathbf{X}$. This is the orthogonal decomposition $\mathbf{H}_{\mathbf{1}_n} + \mathbf{H}_{\mathbf{M}_{\mathbf{1}_n}\mathbf{X}}$ and since the variables are orthogonal, the coefficient β_0 corresponding to the intercept is $\bar{\boldsymbol{y}}$. Having scaled inputs $\mathbf{Z}$ ensures that we penalize equally every covariate and that the model is invariant to change of units for the regressors.

The two methods briefly covered in the course are ridge regression and LASSO. The first one consists of imposing a l_2 penalty on the coefficients, the second a l_1 penalty. The advantage of the former is that the optimization problem is convex and differentiable and the solution can be found using an augmented linear regression. We will solely focus on ridge regression in the sequel.

Our objective function for ridge regression takes the form

$$Q(\beta_0, \boldsymbol{\gamma}) = (\boldsymbol{y} - \beta_0\mathbf{1}_n - \mathbf{Z}\boldsymbol{\gamma})^\top(\boldsymbol{y} - \beta_0\mathbf{1}_n - \mathbf{Z}\boldsymbol{\gamma}) + \lambda\|\boldsymbol{\gamma}\|_2^2.$$

8.1 Bias and variance tradeoff

As we increase the penalty λ, the values of the ridge coefficients are shrunk towards zero. The case $\lambda \to \infty$ gives $\hat{\boldsymbol{\beta}}_{\text{ridge}} = \mathbf{0}_p$, whereas we retrieve the OLS estimator $\hat{\boldsymbol{\beta}}$ when $\lambda = 0$.

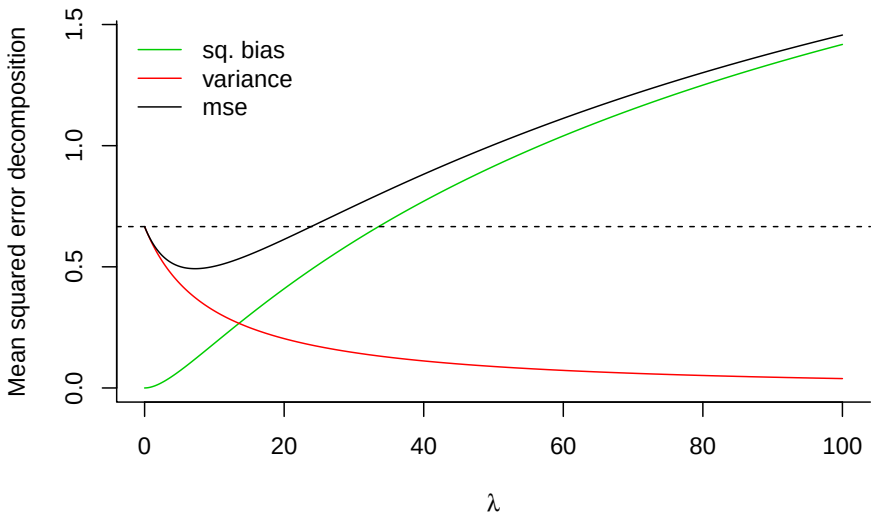
The mean squared error of the ridge estimator is

$$\begin{aligned}\text{MSE}(\hat{\boldsymbol{\beta}}^\lambda_{\text{ridge}}) = \sigma^2\text{tr}\left\{(\mathbf{Z}^\top\mathbf{Z} + \lambda\mathbf{I}_p)^{-1}\mathbf{Z}^\top\mathbf{Z}(\mathbf{Z}^\top\mathbf{Z} + \lambda\mathbf{I}_p)^{-1}\right\} \\ + \boldsymbol{\gamma}^\top\left\{\mathbf{Z}^\top\mathbf{Z}(\mathbf{Z}^\top\mathbf{Z} + \lambda\mathbf{I}_p)^{-1} + \mathbf{I}_p\right\}\left\{(\mathbf{Z}^\top\mathbf{Z} + \lambda\mathbf{I}_p)^{-1}\mathbf{Z}^\top\mathbf{Z} + \mathbf{I}_p\right\}\boldsymbol{\gamma}\end{aligned}$$

If we knew the true data generating mechanism (i.e. the $\boldsymbol{\gamma}$ vector and σ^2), we could compute exactly the mean squared error (MSE) of the model and find the optimal bias-variance tradeoff that minimizes MSE. This is illustrated below in an artificial example. As $\lambda \to \infty$, the bias grows and dominates MSE.

```
# Function to compute MSE of ridge estimator
mse_ridge <- function(gamma, lambda, Z, sigmasq = 1){
  ZtZ <- crossprod(Z)
```

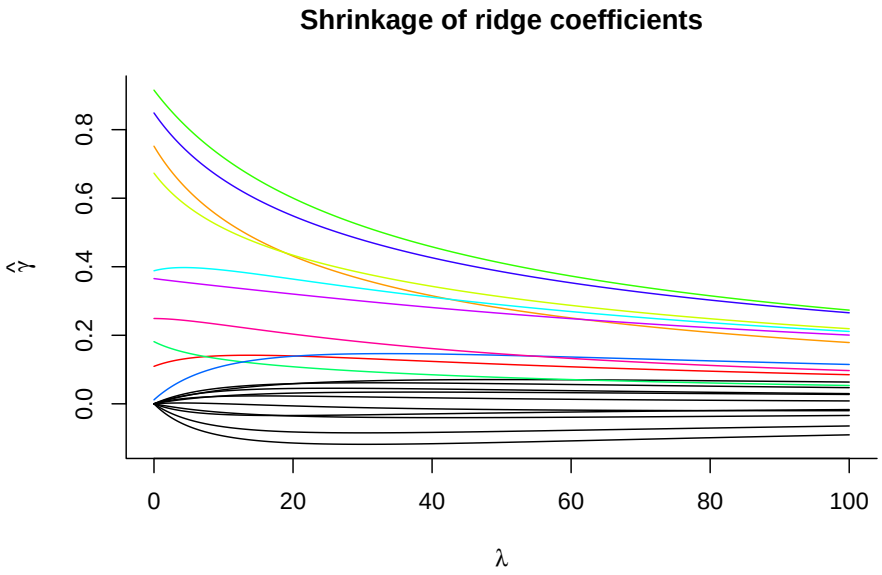
```
  p <- ncol(Z)
  W <- solve(ZtZ + lambda*diag(p))
  bias <- c((W %*% ZtZ - diag(p)) %*% gamma)
  varia <- sigmasq * diag( crossprod(Z %*% W))
  list(bias = bias, variance = varia, mse = sum(bias^2 + varia))
}
set.seed(9876)
# Create fake data
Z <- matrix(rnorm(n = 20L*50L, mean = 0, sd = 1), ncol = 20L)
# Center and renormalize Z
Z <- apply(Z, 2, scale)
# Create coefficient vector
gamma <- c(rep(0, 10L), runif(10L))

# Create sequence of lambda and matrix to store results
nlam <- 401L
lambda_seq <- seq(0, 100, length = nlam)
mse <- matrix(0, nrow = nlam, ncol = 3)
gammahat <- matrix(0, nrow = nlam, ncol = ncol(Z))
for(i in 1:nlam){
  # evaluate bias + variance for each lambda
  mse_i <- mse_ridge(gamma = gamma, lambda = lambda_seq[i], Z = Z)
  gammahat[i,] <- gamma + mse_i$bias
  mse[i,1] <-  sum(mse_i$bias^2)
  mse[i,2] <-  sum(mse_i$variance)
  mse[i,3] <- mse_i$mse
}
# Plot the results as a function of lambda
matplot(lambda_seq, mse, type = "l", lty = 1,
        bty = "l", xlab = expression(lambda), col = 3:1,
        ylab = "Mean squared error decomposition")
abline(h = mse[1,3], lty = 2)
legend(x = "topleft", legend = c("sq. bias", "variance", "mse"),
       col = 3:1, lty = 1, bty = "n")
```



```
# Compute the value of lambda that minimizes MSE
lambdaopt <- optimize(f = function(x){
  mse_ridge(gamma = gamma, lambda = x, Z = Z)$mse
  }, interval = c(0, 30))$minimum
lambdaopt
```

```
## [1] 7.227898
```

We can also look at the path of coefficient values $\hat{\boldsymbol{\gamma}}^{\lambda}_{\text{ridge}}$ as a function of λ:



While the overall vector of coefficients are shrunk towards zero, the set of 10 first coefficients $\boldsymbol{\gamma}$, which are exactly zero, stabilize around another value. Note that if we increase the penalty, from λ_1 to λ_2 with $\lambda_1 < \lambda_2$, this **does not** necessarily imply that individual coefficient estimates decrease, i.e. $|\hat{\beta}_j(\lambda_1)| \not\leq |\hat{\beta}_j(\lambda_2)|$ even if $\lambda_1 < \lambda_2$.

The coefficients $\hat{\boldsymbol{\gamma}}^{\lambda}$ can be computed using an augmented linear regression, with response $(\boldsymbol{y}, \mathbf{0}_p)$ and regressor $[\mathbf{Z}^\top, \lambda^{1/2}\mathbf{I}_p]$. Alternatively,

$$\hat{\boldsymbol{\gamma}}^{\lambda} = (\mathbf{Z}^\top\mathbf{Z} + \lambda\mathbf{I}_p)^{-1}\mathbf{Z}^\top\boldsymbol{y}.$$

We can also use the singular value decomposition of $\mathbf{Z} = \mathbf{U}\mathbf{D}\mathbf{V}^\top$ to compute the coefficients

$$\hat{\boldsymbol{\gamma}} = \sum_{j=1}^{p} \frac{d_j}{d_j^2 + \lambda}\mathbf{u}_j^\top\boldsymbol{y}\mathbf{v}_j,$$

where $\mathbf{u}_j$ and $\mathbf{v}_j$ are the jth column of $\mathbf{U}$ and $\mathbf{V}$, respectively. This is most useful for cross-validation where we want to change the value of λ repeatedly, as the SVD of $\mathbf{Z}$ won't change.

```
# Create response vector from model
y <- Z %*% gamma + rnorm(nrow(Z))
# Compute ridge coefficients
ridge_lm_coef <- function(y, Z, lambda){
 Z <- scale(Z)
 c(solve(crossprod(Z) + lambda*diag(ncol(Z))) %*% crossprod(Z, y - mean(y)))
}
lambda0 <- 2
#Compare coefficients obtained from fitting using augmented regression
```

```
augmy <- c(y - mean(y), rep(0, ncol(Z)))
augmZ <- rbind(Z, sqrt(lambda0)*diag(ncol(Z)))
augmlm <- lm(augmy ~ -1 + augmZ)
isTRUE(all.equal(as.vector(augmlm$coef),
      ridge_lm_coef(y = y, Z = Z, lambda = lambda0)))
```

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

8.2 Cross-validation

Denote the ridge estimator $\hat{\boldsymbol{\beta}}^{\lambda}$ for a given penalty parameter λ. The smoothing matrix, as given in the course notes, is

$$\mathbf{S}_\lambda = \mathbf{Z}(\mathbf{Z}^\top\mathbf{Z} + \lambda \mathbf{I}_p)^{-1}\mathbf{Z}^\top;$$

its trace is $\text{tr}(\mathbf{S}_\lambda) = \sum_{j=1}^{p} d_j^2/(d_j^2+\lambda)$, where d_j is the jth singular value of $\mathbf{Z}$. The smoothing matrix $\mathbf{S}_\lambda$ is not a projection matrix; its eigenvalues lie in $(0,1)$.

Similar calculations than those used to derive the leave-one-out cross validation residual errors of the PRESS statistic lead to the formula

$$\text{CV}_\lambda = \sum_{i=1}^{n} e_{-i}^2(\lambda) = \sum_{i=1}^{n} (y_i - \bar{y} - \mathbf{z}_i\hat{\boldsymbol{\gamma}}^{\lambda}_{-i})^2 = \sum_{i=1}^{n} \frac{(y_i - \bar{y} - \mathbf{z}_i\hat{\boldsymbol{\gamma}}^{\lambda})^2}{1-\mathbf{S}_{\lambda\, ii}},$$

where $\mathbf{z}_i$ is the ith row of $\mathbf{Z}$. Rather than compute $\mathbf{S}_\lambda$ and its diagonal elements, one can resort to the convenient generalized cross-validation approximation

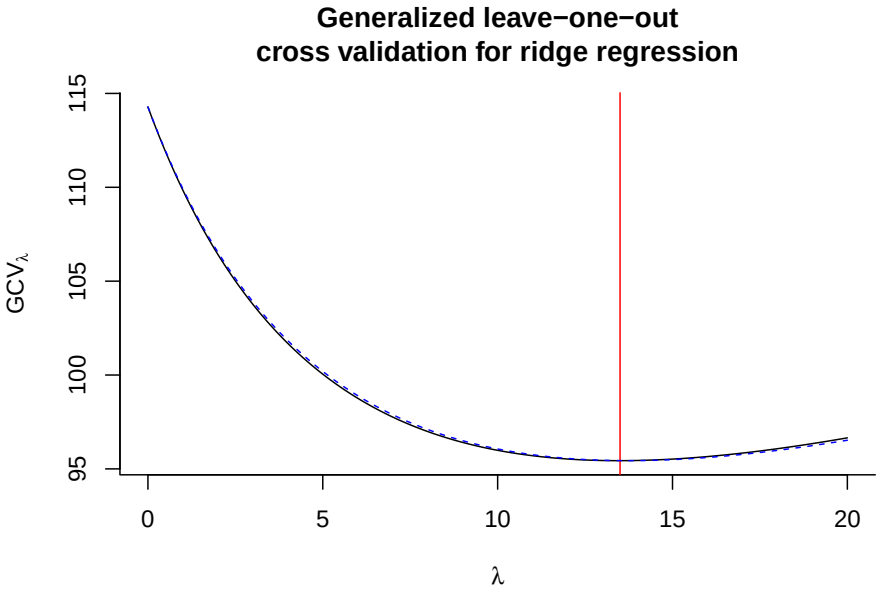
$$\text{GCV}_\lambda = \sum_{i=1}^{n} \frac{(y_i - \bar{y} - \mathbf{z}_i\hat{\boldsymbol{\gamma}}^{\lambda})^2}{1-\text{tr}(\mathbf{S}_\lambda)/n}$$

and the latter is readily computed.

```
nlam <- 201L
lambda_seq <- seq(0, 20, length = nlam)
svdZ <- svd(Z)
n <- nrow(Z); p <- ncol(Z)
#Each column is u_j^Tyv_j
uyv <- sapply(1:p, function(j){t(svdZ$u[,j]) %*% y %*% svdZ$v[,j]})
gcv <- rep(0, nlam)
yc <- y - mean(y)
for(i in 1:nlam){
  # Compute GCV - trace of smoother + RSS
  traceS <- sum(svdZ$d^2/(svdZ$d^2+lambda_seq[i]))
  gcv[i] <- sum((yc - Z %*% colSums(c(svdZ$d/(svdZ$d^2 + lambda_seq[i])) *
                                    t(uyv)))^2)/(1-traceS/n)^2
}
# Plot GCV curve against lambda
plot(lambda_seq, gcv, type = "l", bty = "l",
     main = "Generalized leave-one-out\ncross validation for ridge regression",
     ylab = expression(GCV[lambda]), xlab = expression(lambda))
abline(v = lambda_seq[which.min(gcv)], col = 2)
lambda_seq[which.min(gcv)]
```

```
## [1] 13.5
```

```
# Compare results with MASS
library(MASS)
fit_ridge <- lm.ridge(y ~ Z, lambda = lambda_seq)
# GCV returned by lm.ridge is divided by n^2
lines(lambda_seq, fit_ridge$GCV*n^2, lty = 2, col = "blue")
```




```
select(fit_ridge)
```

```
## modified HKB estimator is 8.577316
## modified L-W estimator is 7.881568
## smallest value of GCV  at 13.7
```

Note that in this case, the optimal value of λ found is higher than the theoretical optimum. In practice, we may prefer K-fold cross-validation to leave-one-out cross validation.

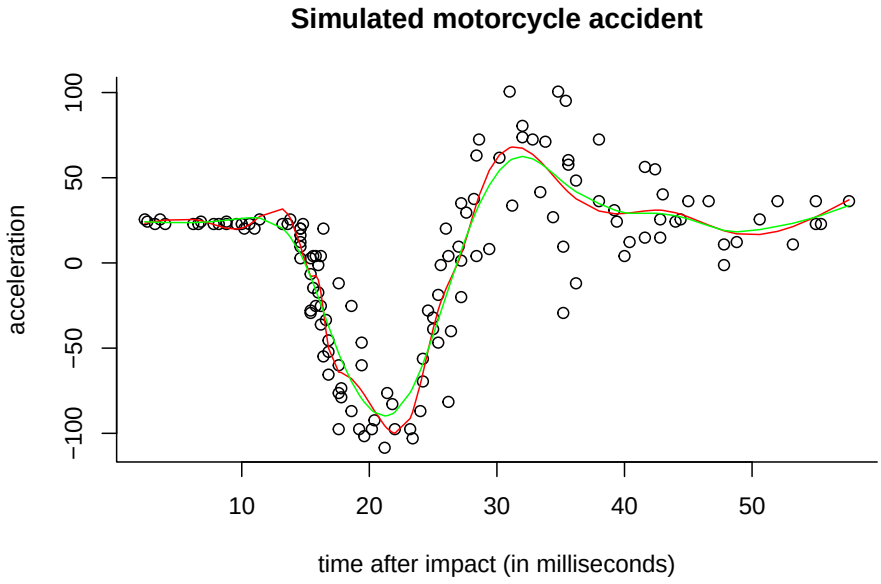
Chapter 9

Splines

The following code provides functions to compute manually a cubic spline and returns the penalty function. In **R**, one would rather use functions that compute efficiently the generalized cross-validation criterion and return the best smoothing or regression spline.

The package `splines` contains `bs` for cubic regression splines and `ns` for natural regression splines. Smoothing splines can be obtained by a call to `smooth.spline`. The following illustrates a regression spline in which a number of knots (here roughly $n/10$) are chosen equispaced on the distribution scale, based on the quantiles of the covariate vector.

```
library(splines)
data(mcycle, package = "MASS")
# Center observations
x <- mcycle$times
y <- mcycle$accel - mean(mcycle$accel)
n <- length(x)
df <- floor(n/10)
# Compute number of knots
m <- df + 4
knots <-  quantile(x, prob = (1:m)/(m+1))

#With regression natural cubic spline, equispaced knots on quantile scale
plot(y~x, xlab = "time after impact (in milliseconds)", ylab = "acceleration",
     main = "Simulated motorcycle accident", bty = "l")
lines(x, fitted(lm(y~ns(x = x, knots = knots))), col = "red")
#Smoothing spline, using GCV
spl <- smooth.spline(x = x, y = y, cv = FALSE, all.knots = TRUE)
lines(x, fitted(spl), col = "green")
```



example-1.bb

The fit from the regression spline differs from that of the smoothing spline. The former uses only 18 basis functions and therefore can be fitted using `lm`, whereas the smoothing spline uses 94 basis functions (this is lower than n because there are ties in the `x` vector).

Below is sample code to obtain the $\mathbf{B}$ and $\mathbf{\Omega}$ matrices, corresponding to the basis functions and the associated penalty.

```
# You need the fda package installed
# install.packages("fda")
# Function returns B and Omega matrices
smoothsplineb <- function(x){
  require(fda)
  breaks <- sort(unique(x))
  basisfd <- create.bspline.basis(norder = 4, breaks = breaks)
  B <- bsplineS(x, breaks = breaks, norder = 4, nderiv = 0)
  penmat <- bsplinepen(basisfd, Lfd = 2)
  list(basis = B, penmat = penmat)
}

# Compute ridge regression coefficients manually
lmridge <- function(y, X, O = diag(length(y)), lambda){
  as.vector(solve(crossprod(X) + lambda*O, crossprod(X, y)))
}
```

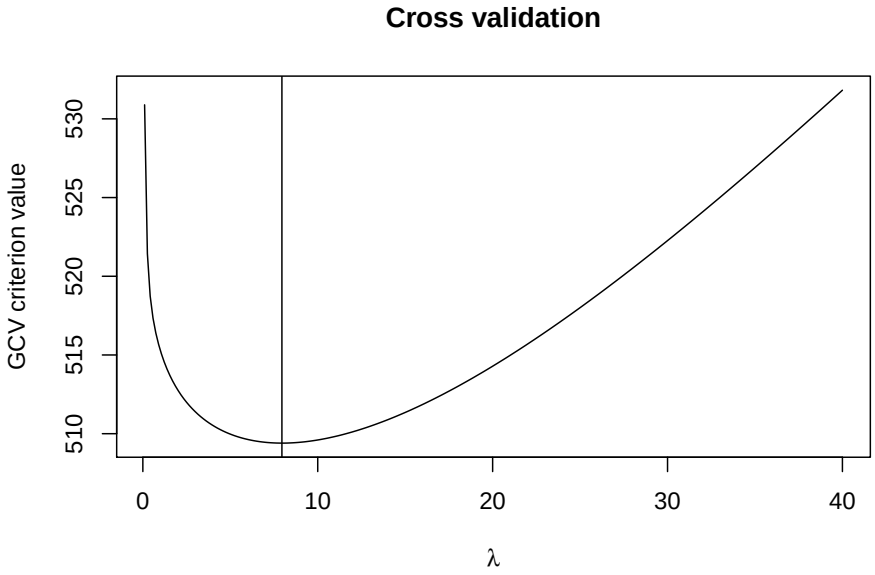
For the `mcycle` data, the following lines of code return those matrices.

```
smoothspline_mcycle <- smoothsplineb(x = x)
B <- smoothspline_mcycle$basis
O <- smoothspline_mcycle$penmat
```

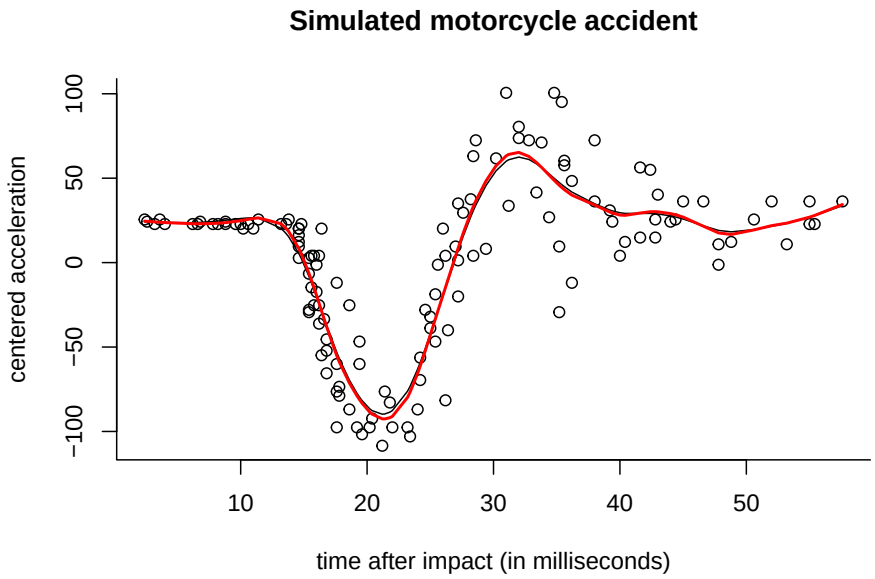
9.1 Solutions

9.1.1 Exercise 12.4 Smoothing splines

```
# Evaluate GCV at a grid of lambda
lambdas <- seq(0.1, 40, length = 250L)
# Container for GCV criterion
gcv <- rep(0, length(lambdas))
# Loop over cases
for(i in 1:length(lambdas)){
  # Compute smoothing matrix
  Sm <- B %*% solve(crossprod(B) + lambdas[i]*O) %*% t(B)
  # Compute ridge regression coefficients
  coefs <- lmridge(y = y, X = B, O = O, lambda = lambdas[i])
  # Compute GCV criterion
  gcv[i] <- c(crossprod(y - B %*% coefs)/(1-mean(diag(Sm))))/nrow(B)
}
# Plot GCV
plot(lambdas, gcv, type = "l", xlab = expression(lambda),
     ylab = "GCV criterion value", main = "Cross validation")
abline(v = lambdas[which.min(gcv)])
```




```
# Plot data
plot(y ~ x, xlab = "time after impact (in milliseconds)", ylab = "centered acceleration",
     main = "Simulated motorcycle accident", bty = "l")
# Undersmoothing
#lines(x, B %*% lmridge(y = y, X = B, O = O, lambda = 0.01), col = "green")
# Oversmoothing
#lines(x, B %*% lmridge(y = y, X = B, O = O, lambda = 500), col = "blue")
lines(predict(smooth.spline(y = y, x = x, all.knots = TRUE)))
fitted_opt <- B %*% lmridge(y = y, X = B, O = O,
                    lambda = lambdas[which.min(gcv)])
lines(x, fitted_opt , col = "red", lwd = 2)
```


Simulated motorcycle accident
centered acceleration
100
50
0
−50
−100
10
20
30
40
50
time after impact (in milliseconds)