lineaRmodels

Léo Belzile

version of 2018-09-30

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.

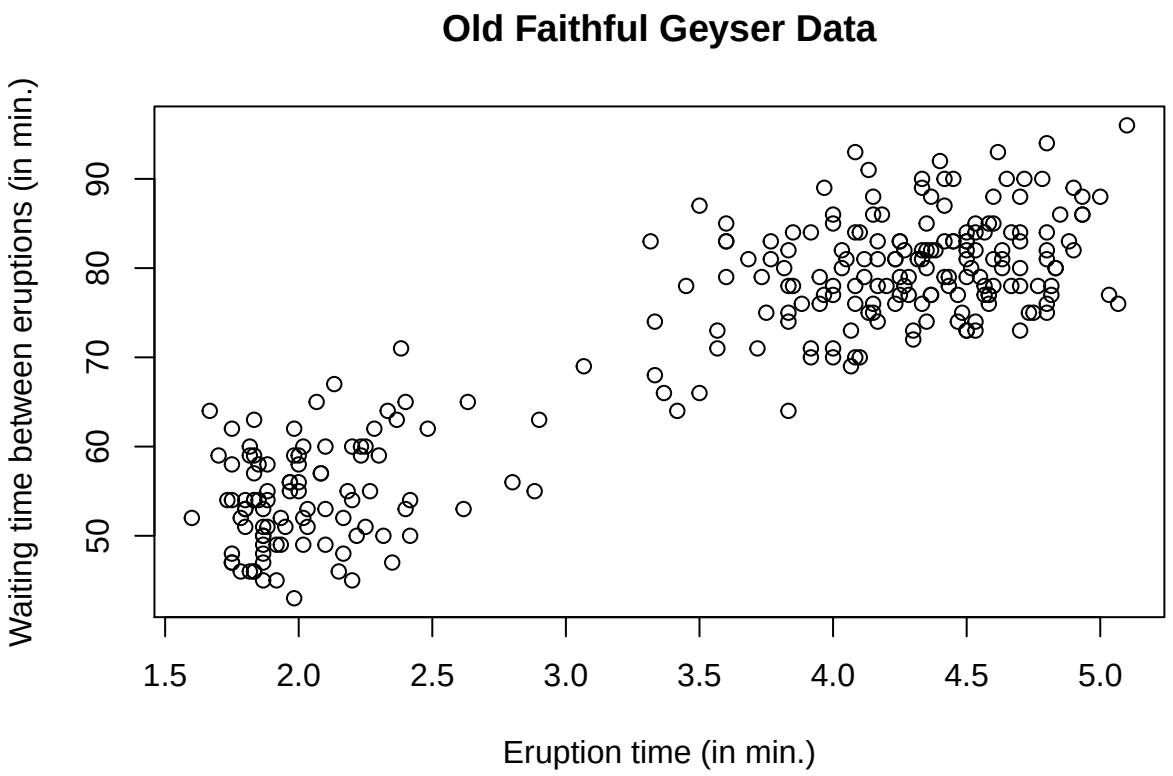
Once `plot` has been called, you can add additional observations as points (lines) to the graph using `point` (`lines`) in place of `plot`. If you want to add a line (horizontal, vertical, or with known intercept and slope), use the function `abline`.

Other functions worth mentioning at this stage:

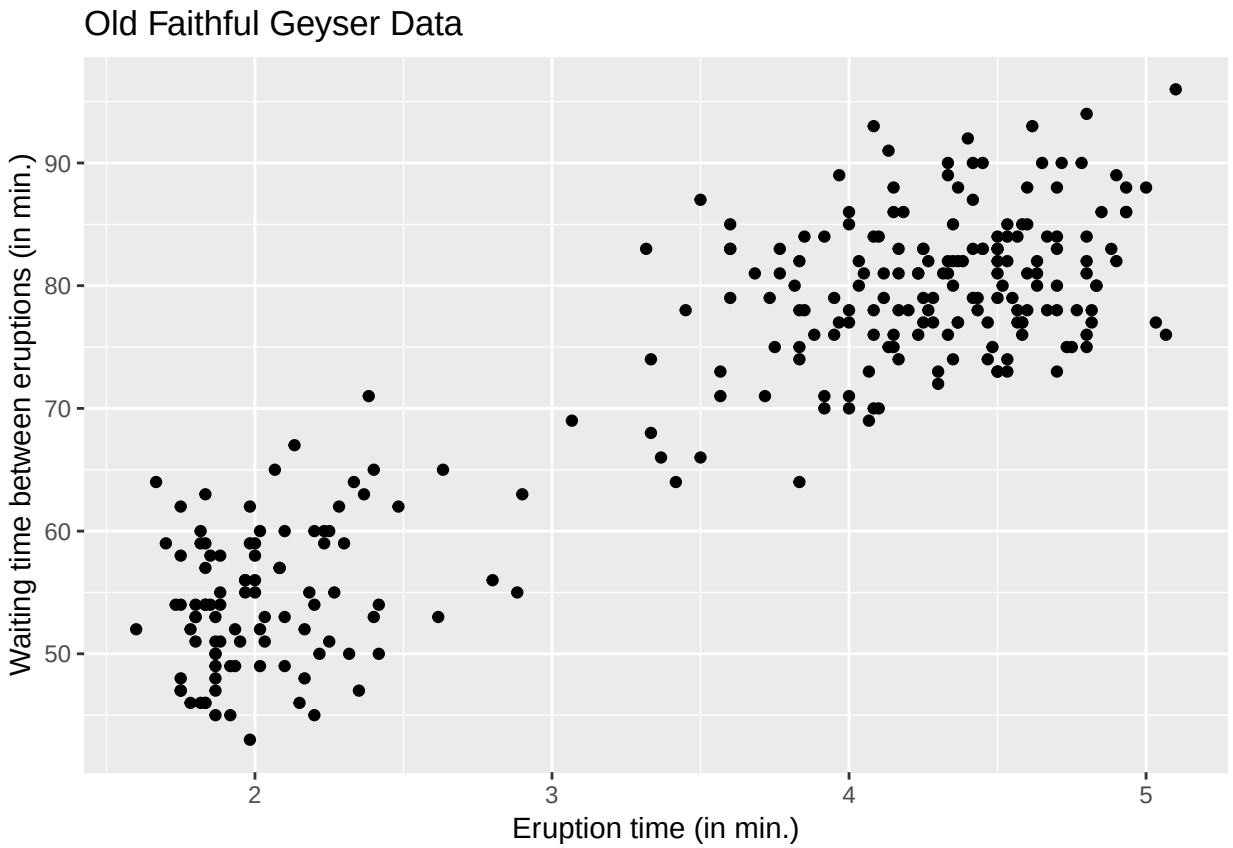
- `boxplot` creates a box-and-whiskers plot
- `hist` creates an histogram, either on frequency or probability scale (option `freq = FALSE`). `breaks` control the number of bins. `rug` adds lines below the graph indicating the value of the observations.
- `pairs` creates a matrix of scatterplots, akin 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 $\mathbf{H}_\mathbf{X} \equiv \mathbf{X}(\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top$ is the orthogonal projection matrix onto 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. We can verify the properties of $\mathbf{H}_\mathbf{X}$ 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.

```r
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
```

```r
#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
```

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

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

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

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

```r
#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
```

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

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

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.

1.3.2 Oblique projections (exercise 1.4)

Suppose that $\mathsf{span}(\mathbf{X}) \neq \mathsf{span}(\mathbf{W})$, 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 $\mathrm{im}(\mathbf{P}) = \mathsf{span}(\mathbf{X})$, but $\mathrm{im}(\mathbf{I} - \mathbf{P}) = \mathsf{span}(\mathbf{W}^\perp)$. This fact is illustrated below.

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

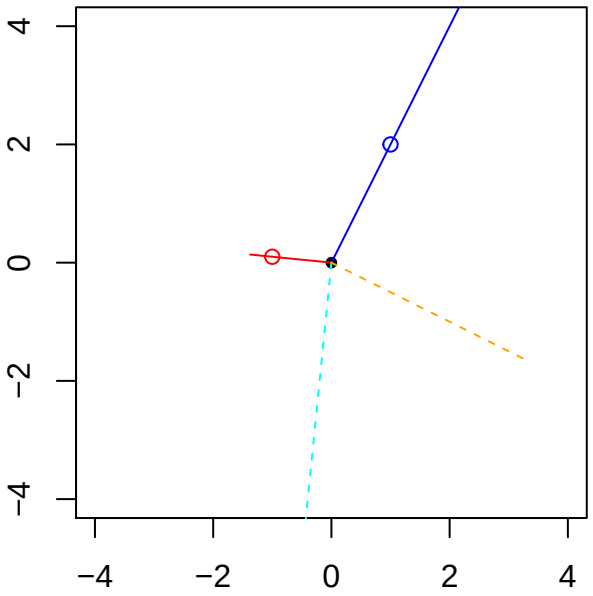
```
#Create two vectors (non-parallel)
x <- c(1, 2)
w <- c(-1, 0.1)
#Create oblique projection matrix
P <- x %*% solve(t(w) %*% x) %*% t(w)

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 $\mathbf{I}_2 - \mathbf{P} \perp \mathbf{P}^\top$.



1.4 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 $\mathcal{V} = \mathsf{span}(\mathbf{X})$, then

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

Since $\mathbf{X} : \mathbb{R}^n \to \mathbb{R}^p$, $\mathbf{H}_\mathbf{X}$ has rank p. The orthogonal complement $\mathbf{M}_\mathbf{X} \equiv \mathbf{I}_n - \mathbf{H}_\mathbf{X}$ projects onto $\mathsf{span}^\perp(\mathbf{X})$.

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 $\mathbf{H_X}$ 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).

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

2.1.1 Normal equations

The following simply illustrates what has been derived in Exercise series 2. You will probably never use these commands, as **R** has devoted functions that are coded more efficiently. We can compute first the ordinary least square estimates using the formula $\hat{\boldsymbol{\beta}} = (\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top\boldsymbol{y}$.

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

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

```
betahat <- as.vector(solve(XtX, Xty))
#same as betahat <- solve(t(X) %*% X) %*% t(X) %*% y
```

2.1.2 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 $\mathbf{X}^\top\mathbf{X}$.

```
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
betahat_svd <- c(svdX$v %*%  diag(1/svdX$d) %*% t(svdX$u) %*% y)
all.equal(betahat_svd, betahat)
```

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

2.1.3 QR decomposition

R uses a QR-decomposition to calculate the OLS. 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,

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

```
qrX <- qr(X)
Q1 <- qr.Q(qrX)
R <- qr.R(qrX)
# Compute betahat from QR
betahat_qr1 <- qr.coef(qrX, y) #using built-in function
betahat_qr2 <- c(backsolve(R, t(Q1) %*% y)) #manually
all.equal(betahat, betahat_qr1, check.attributes = FALSE)
```

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

```
all.equal(betahat, betahat_qr2, check.attributes = FALSE)
```

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

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

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

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

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

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

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

2.2 Parameter estimation

We are now ready to fit a simple linear model with an intercept and a linear effect for the weight,

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

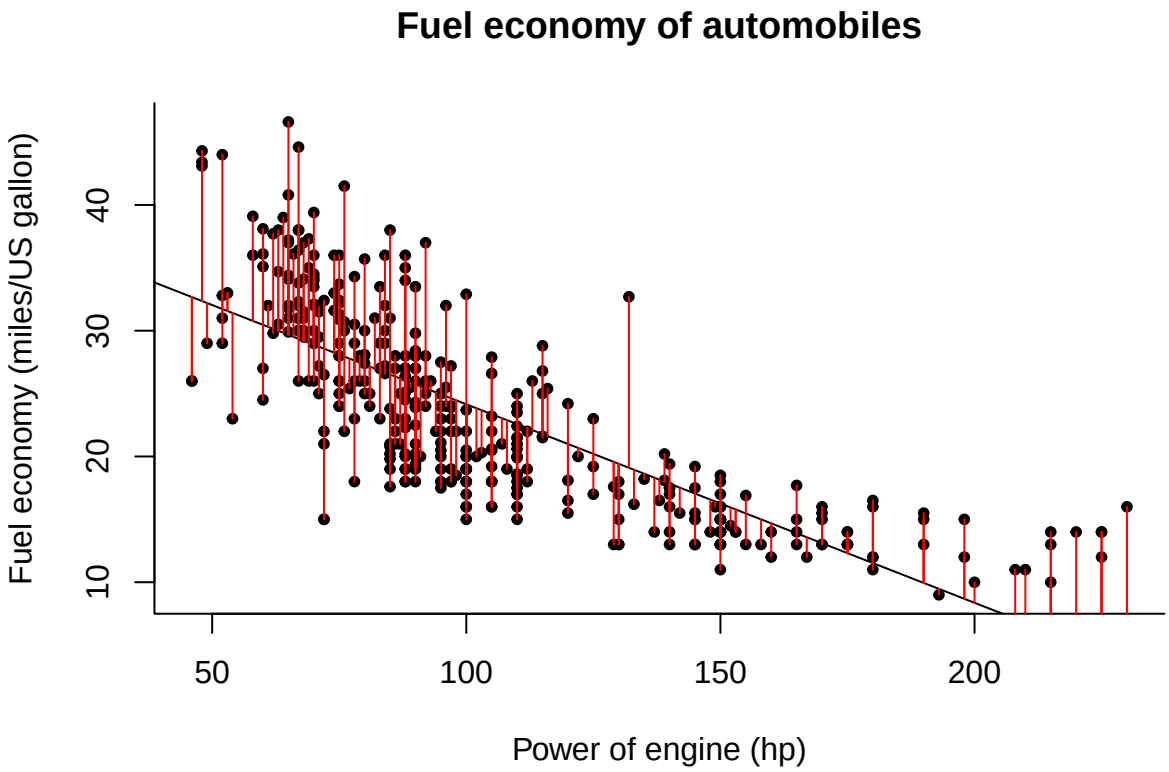
We form the design matrix $(\mathbf{1}_n^\top, \texttt{hp}^\top)^\top$ and the vector of regressand `mpg`, then proceed with calculating the OLS coefficients $\hat{\boldsymbol{\beta}}$, the hat matrix $\mathbf{H}_\mathbf{X}$, the fitted values $\hat{\boldsymbol{y}}$ and the residuals $\boldsymbol{e}$.

```
#Design matrix
hp <- Auto$horsepower
X <- cbind(1, Auto$horsepower)
mpg <- Auto$mpg
#OLS estimates
XtXinv <- solve(crossprod(X))
beta_hat <- c(XtXinv %*% t(X) %*% mpg)
#Form orthogonal projection matrix
```

```
Hmat <- X %*% XtXinv %*% t(X)
#Create residuals and fitted values
fitted <- Hmat %*% mpg
res <- mpg - fitted
fitted <- Hmat %*% mpg
```

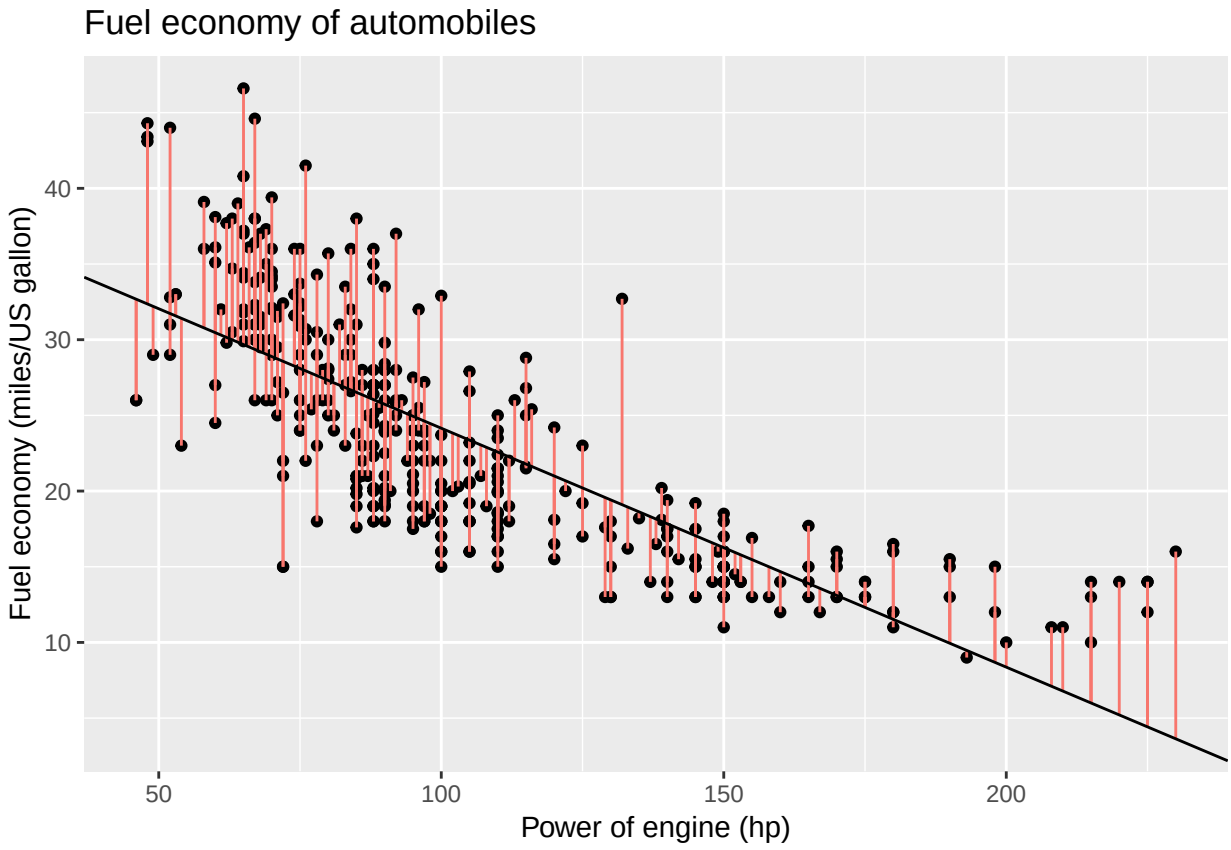
The residuals $\boldsymbol{e} = \boldsymbol{y} - \hat{\boldsymbol{y}}$ can be interpreted as the *vertical* distance between the regression slope and the observation. For each observation y_i, a vertical line at distance e_i is drawn from the prediction $\hat{y}_i$.

```
plot(mpg ~ horsepower,  data = Auto,
     xlab = "Power of engine (hp)",
     ylab = "Fuel economy (miles/US gallon)",
     main = "Fuel economy of automobiles",
     # 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 = hp[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 = hp, 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)
```



2.3 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.

2.4 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 $\mathrm{H}_0 : \beta_i = 0$ 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)
```

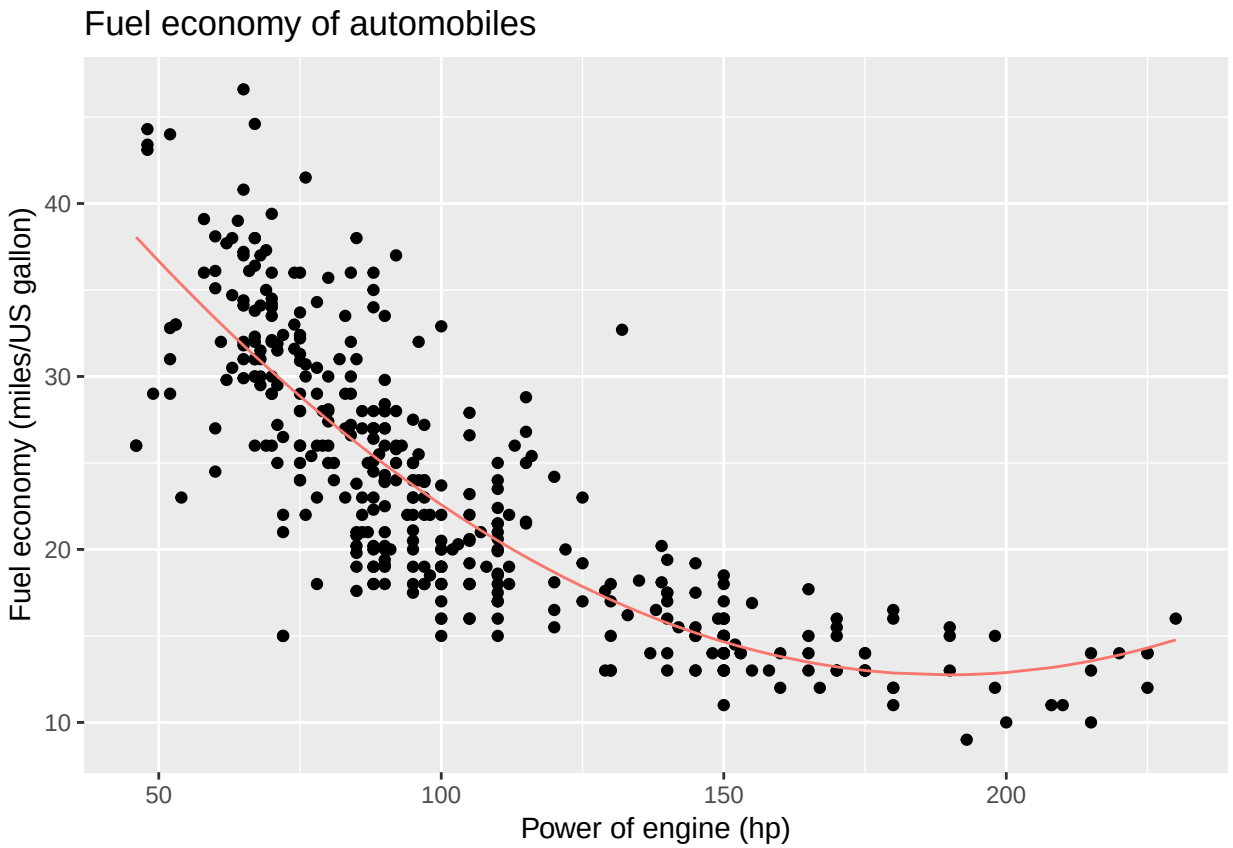
```
##  [1] "call"          "terms"         "residuals"     "coefficients"
##  [5] "aliased"       "sigma"         "df"            "r.squared"
##  [9] "adj.r.squared" "fstatistic"    "cov.unscaled"
```

2.5 The hyperplane of fitted values

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

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

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



This quadratic curve is nothing like an hyperplane! Let $\boldsymbol{y} \equiv$ `mpg`, `x` = `hp` and `z` = `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.

```
## PhantomJS not found. You can install it with webshot::install_phantomjs(). If it is installed, pleas
```

2.6 (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.,

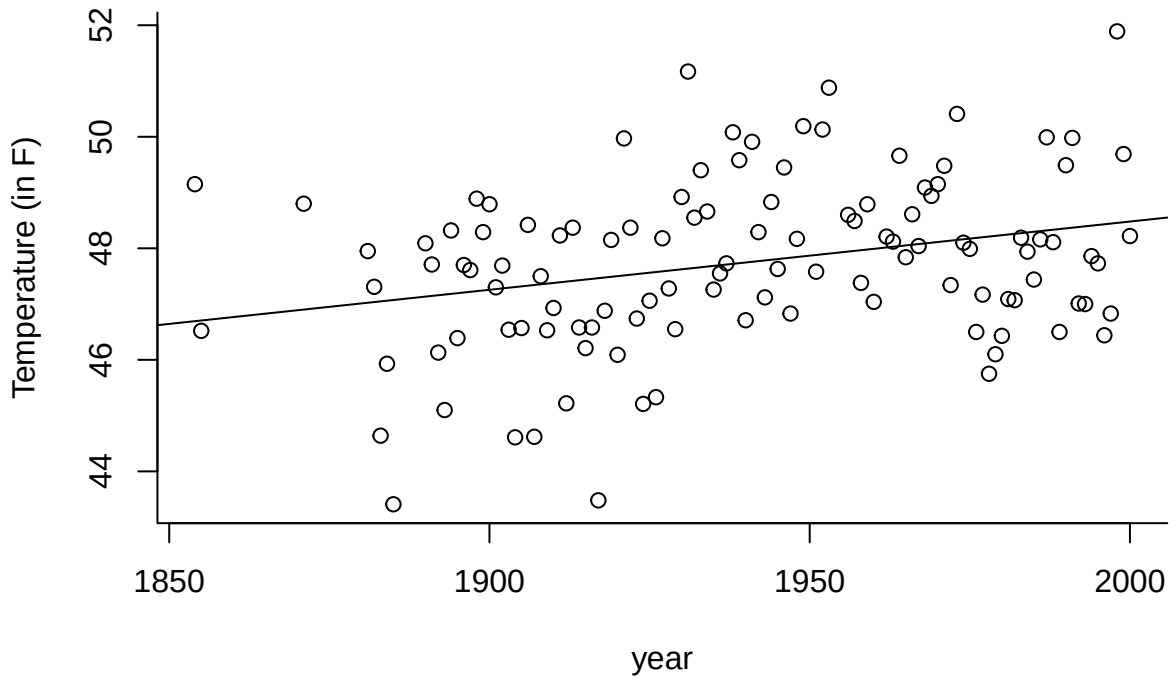
$$R_c^2 = \frac{\|\hat{\boldsymbol{y}} - \bar{y}\mathbf{1}_n\|^2}{\|\boldsymbol{y} - \bar{y}\mathbf{1}_n\|^2} = \frac{\|\hat{\boldsymbol{y}}\|^2 - \|\bar{y}\mathbf{1}_n\|^2}{\|\boldsymbol{y}\|^2 - \|\bar{y}\mathbf{1}_n\|^2}.$$

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

Provided that $\mathbf{1}_n \in \mathsf{span}(\mathbf{X})$, it is obvious that the fitted values $\hat{\boldsymbol{y}}$ are invariant to linear transformations of the covariates $\mathbf{X}$. 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.

2.7 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, $\boldsymbol{e} \perp \hat{by}$.

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 $\|\boldsymbol{y}\|^2 = \|\hat{\boldsymbol{y}}\|^2 + \|\boldsymbol{e}\|^2$.

2.8 Exercises

2.8.1 Prostate cancer dataset

The following questions refer to the dataset `prostate` from the package `ElemStatLearn`. - Briefly describe the dataset. - Look at summaries of `lbph`. What likely value was imputed in places of zeros in `lbph} (before taking the logarithm)? - 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. `- 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 $\hbb$ of the parameters, the fitted values $\hat{\bs{y}}$ and the residuals $\bs{e}$ using the formulas given in class. - Compare the quantities you obtained with the output of the function` lm. `- Add the fitted regression line to the scatterplot of` lcavol `against` lpsa. `- Interpret the changes in cancer volume (not the log cancer volume), including any units in your interpretations. - Obtain the orthogonal projection matrix $\bf{H}_\mX$ and the OLS coefficients $\hbb$ using a SVD decomposition of $\mX$ (svd). - Compute the R^2_c 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`‘?`

Chapter 3

Frisch–Waugh–Lovell theorem

This result dates back to the work of Frisch, R. and F. Waugh (1933)[1] and of M. Lovell (1963)[2]. 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*

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

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

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

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

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. □

3.1 Examples

We have seen last week the centered R_c^2; the latter is equivalent to the coefficient of determination from the FWL regression

$$\mathbf{M}_{\mathbf{1}_n}\boldsymbol{y}=\mathbf{M}_{\mathbf{1}_n}\mathbf{X}\boldsymbol{\beta}+\boldsymbol{\varepsilon},$$

that is, after centering regressand and regressors. Note that this regression will have no intercept, because this coefficient would be exactly zero.

To be continued.