Computing for Big Data (BST-262)

Christine Choirat

2017-10-12

Contents

1 Introduction **5**
- 1.1 Prerequisites . . . 5
- 1.2 Rationale . . . 5
- 1.3 Syllabus . . . 5
- 1.4 Evaluation . . . 6
- 1.5 Software tools and packages . . . 6
- 1.6 Datasets . . . 6
- 1.7 Contributing with GitHub . . . 7

2 Basic tools **9**
- 2.1 Command line tools . . . 9
- 2.2 Makefiles . . . 11
- 2.3 Git and GitHub . . . 15

3 Packages **17**
- 3.1 Why? . . . 17
- 3.2 Package structure . . . 17
- 3.3 Building steps . . . 17
- 3.4 Create an R package . . . 18
- 3.5 R packages on GitHub . . . 19
- 3.6 RStudio projects . . . 20
- 3.7 Package workflow example . . . 21
- 3.8 Unit testing . . . 25
- 3.9 Code coverage . . . 26
- 3.10 Continuous integration . . . 26
- 3.11 Vignettes . . . 26

4 Optimization **27**
- 4.1 Measuring performance . . . 27
- 4.2 Improving performance . . . 27

5 Databases **29**
- 5.1 Overview . . . 29
- 5.2 SQL . . . 29
- 5.3 noSQL . . . 29
- 5.4 R interfaces . . . 29

6 Big data **31**
- 6.1 Reading big data (that fits in memory) . . . 31
- 6.2 Sampling (can be read, not analyzed easily) . . . 31
- 6.3 Pure R solutions . . . 31
- 6.4 JVM solutions . . . 31

7 Visualization **33**
7.1 Principles of visualization 33
7.2 Maps and GIS 33

Chapter 1

Introduction

1.1 Prerequisites

For BST262 (Computing for Big Data), we assume familiarity with the material covered in BST260 (Introduction to Data Science).

We will use R to present concepts that are mostly language-agnostic. We could have used Python, as in BST261 (Data Science II).

1.2 Rationale

1. Available data grows at a much faster rate than available computing capacity.
2. Statistical software programs such as R were not designed to handle datasets of massive size.

1.3 Syllabus

Week 1 - Basic tools

- Lecture 1. Unix scripting, make
- Lecture 2. Version control: Git and GitHub (guest lecture: Ista Zhan)

Week 2 - Creating and maintaining R packages

- Lecture 3. Rationale, package structure, available tools
- Lecture 4. Basics of software engineering: unit testing, code coverage, continuous integration

Week 3 - Software optimization

- Lecture 5. Measuring performance: profiling and benchmarking tools
- Lecture 6. Improving performance: an introduction to C/C++, Rcpp

Week 4 – Databases

- Lecture 7. Overview of SQL (SQLite, PostgreSQL) and noSQL databases (HBase, MongoDB, Cassandra, BigTable, …)
- Lecture 8. R database interfaces (in particular through dplyr)

Week 5 - Analyzing data that does not fit in memory

- Lecture 9. Pure R solutions (sampling, `ff` and `bigmemory`, other interpreters). JVM solutions (h20, Spark)
- Lecture 10. An introduction to parallel computing; clusters and cloud computing. "Divide and Conquer" (MapReduce approaches)

Week 6 – Visualization

- Lecture 11. Principles of visualization (guest lecture: James Honaker)
- Lecture 12. Maps and GIS: principles of GIS, using R as a GIS, PostGIS

Weeks 7 & 8 - Guest lectures (order and precise schedule TBD)

- Software project management (Danny Brooke)
- R and Spark (Ellen Kraffmiller and Robert Treacy)
- Advanced GIS and remote sensing (TBD)
- Cluster architecture (William J. Horka)

1.4 Evaluation

Grades will be based on **two mandatory problem sets**. Each problem set will correspond to 50% (= 50 points) of the final grade. The first problem set will be available by the end of week 3 and the second problem set by the end of week 6.

You will be required to submit problem set solutions within two weeks. Grades, and feedback when appropriate, will be returned two weeks after submission.

You will submit a markdown document that combines commented code for data analysis and detailed and structured explanations of the algorithms and software tools that you used.

1.5 Software tools and packages

We will mostly use R in this course. Some examples will be run in Python.

In general, we will use free and open-source software programs such as PostgreSQL / PostGIS or Spark.

1.6 Datasets

We have collected datasets to illustrate concepts. They are hosted on a Dropbox folder.

1.6.1 MovieLens

MovieLens by Harper and Konstan (2015, https://grouplens.org/datasets/movielens/) collects datasets from the website https://movielens.org/.

There are datasets of different sizes. We will use:

1. Small (1MB): https://grouplens.org/datasets/movielens/latest/
2. Benchmark (~190MB zipped): https://grouplens.org/datasets/movielens/20m/

1.6.2 Airlines data

The airlines dataset comes from the U.S. Department of Transportation and were used in the 2009 Data Expo of the American Statistical Association (ASA).

We will use a version curated by h20o: https://github.com/h2oai/h2o-2/wiki/Hacking-Airline-DataSet-with-H2O.

1.6.3 Insurance claims

Claims data contain Protected Health Information (PHI). There are strong privacy restrictions to store, use and share this type of data.

We will use synthetic data (Sample 1) from the Centers for Medicare and Medicaid Services (CMS).

1.6.4 Census

Census data is commonly merged with administrative claims data such as Medicare. We will use data from the Census Bureau.

1.6.5 $PM_{2.5}$ exposure

We will use $PM_{2.5}$ exposure data from the EPA Air Quality System (AQS) to illustrate GIS linkage concepts.

1.6.6 Methylation

If there is enough interest, we might present methylation examples.

1.7 Contributing with GitHub

If you have suggestions, you can open a GitHub issue at https://github.com/cchoirat/bigdata17/issues.

If you want to contribute, we welcome pull requests.

Chapter 2

Basic tools

In this Chapter, we present basic tools that will be important when interacting with big data systems, most importantly the command-line interface (CLI) in a Unix shell and several utilities (`less`, `awk`, `vi` and `make`).

2.1 Command line tools

We assume some familiarity with the Unix shell, for example as in http://swcarpentry.github.io/shell-novice/.

We also assume that you have access to a shell, either because you use Linux or OS X or because you have the right tools on Windows (for example Cygwin or the Bash shell in Windows 10).

2.1.1 Why use the command line?

- Batch processing
- Cluster and cloud computing

2.1.2 Basic Unix commands

2.1.3 Useful tools

2.1.3.1 `less`

2.1.3.2 `awk`

2.1.3.3 `vi`

2.1.4 Example

Let's apply some of the techniques described in Blackwell and Sen (2012) on Fisher's Iris data set saved in tab-delimited format. Of course, it is a small dataset easily processed with R:

```
iris <- read.table("~/Dropbox/Data17/iris/iris.tab")
head(iris, n = 5)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
```

In a shell, we can use:

```
head -n 6 ~/Dropbox/Data17/iris/iris.tab
```

```
## "Sepal.Length"   "Sepal.Width"   "Petal.Length"  "Petal.Width"   "Species"
## "1"  5.1 3.5 1.4 0.2 "setosa"
## "2"  4.9 3   1.4 0.2 "setosa"
## "3"  4.7 3.2 1.3 0.2 "setosa"
## "4"  4.6 3.1 1.5 0.2 "setosa"
## "5"  5   3.6 1.4 0.2 "setosa"
```

Suppose that we only need to select two variables in our model, `Sepal.Length` and `Species`. In R, we can use:

```
iris_subset <- iris[, c("Sepal.Length", "Species")]
```

or

```
iris_subset <- iris[, c(1, 5)]
head(iris_subset)
```

```
##   Sepal.Length Species
## 1          5.1  setosa
## 2          4.9  setosa
## 3          4.7  setosa
## 4          4.6  setosa
## 5          5.0  setosa
## 6          5.4  setosa
```

With the tidyverse, we can use *pipes*. The `%>%` operator allows for performing chained operations.

```
suppressMessages(library(dplyr))

iris %>%
  select(1, 5) %>%
  head()
```

```
##   Sepal.Length Species
## 1          5.1  setosa
## 2          4.9  setosa
## 3          4.7  setosa
## 4          4.6  setosa
## 5          5.0  setosa
## 6          5.4  setosa
```

In a shell, the pipe operator to combine shell commands is `|` and we can use:

```
cut -f 1,5 ~/Dropbox/Data17/iris/iris.tab | head -n 7
```

```
## "Sepal.Length"   "Species"
## "1"  0.2
## "2"  0.2
## "3"  0.2
```

```
## "4"  0.2
## "5"  0.2
## "6"  0.4
```

To keep observations with "Sepal.Length" greater than 5:

```
iris %>%
  filter(Sepal.Length > 5) %>%
  head()
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          5.4         3.9          1.7         0.4  setosa
## 3          5.4         3.7          1.5         0.2  setosa
## 4          5.8         4.0          1.2         0.2  setosa
## 5          5.7         4.4          1.5         0.4  setosa
## 6          5.4         3.9          1.3         0.4  setosa
```

In the shell, we can use the `AWK` programming language. We start from row `NR` 2 (we could start from row 1, it contains variable names) and select rows such that the second variable (`Sepal.Length`) is greater than 5.

```
awk 'NR == 2 || $2 > 5' ~/Dropbox/Data17/iris/iris.tab | head
```

```
## "1"  5.1 3.5 1.4 0.2 "setosa"
## "6"  5.4 3.9 1.7 0.4 "setosa"
## "11" 5.4 3.7 1.5 0.2 "setosa"
## "15" 5.8 4   1.2 0.2 "setosa"
## "16" 5.7 4.4 1.5 0.4 "setosa"
## "17" 5.4 3.9 1.3 0.4 "setosa"
## "18" 5.1 3.5 1.4 0.3 "setosa"
## "19" 5.7 3.8 1.7 0.3 "setosa"
## "20" 5.1 3.8 1.5 0.3 "setosa"
## "21" 5.4 3.4 1.7 0.2 "setosa"
```

Exercise 2.1. The iris dataset is also saved in .csv format at `~/Dropbox/Data17/iris/iris.csv`. Use `AWK` and `tail` to select the last 5 observations where `Sepal.Width` is larger than 3.5 and `Petal.Length` is smaller than 1.5.

2.2 Makefiles

`make` is a tool that helps put all the pieces of an analytic workflow together:

- data retrieving
- data cleaning
- analysis
- graphs
- reports
- ...

Dependency management

2.2.1 Simulate data in R

```
set.seed(123)
```

File simulate_data.R

```r
# set.seed(123)
N <- 1000 # sample size

X1 <- rpois(n = N, lambda = 50)
X2 <- 10 + rbinom(n = N, prob = 0.8, size = 1)
Y <- 10 + 3 * X1 + -5 * X2 + 3 * rnorm(n = N)

write.csv(data.frame(Y = Y, X1 = X1, X2 = X2),
          "sample_data.csv", row.names = FALSE)
```

```r
head(data.frame(Y = Y, X1 = X1, X2 = X2))
```

```
##           Y X1 X2
## 1  88.74430 46 11
## 2 125.77081 58 11
## 3  70.76396 38 10
## 4 110.32157 50 10
## 5 145.79546 62 11
## 6 109.45403 53 11
```

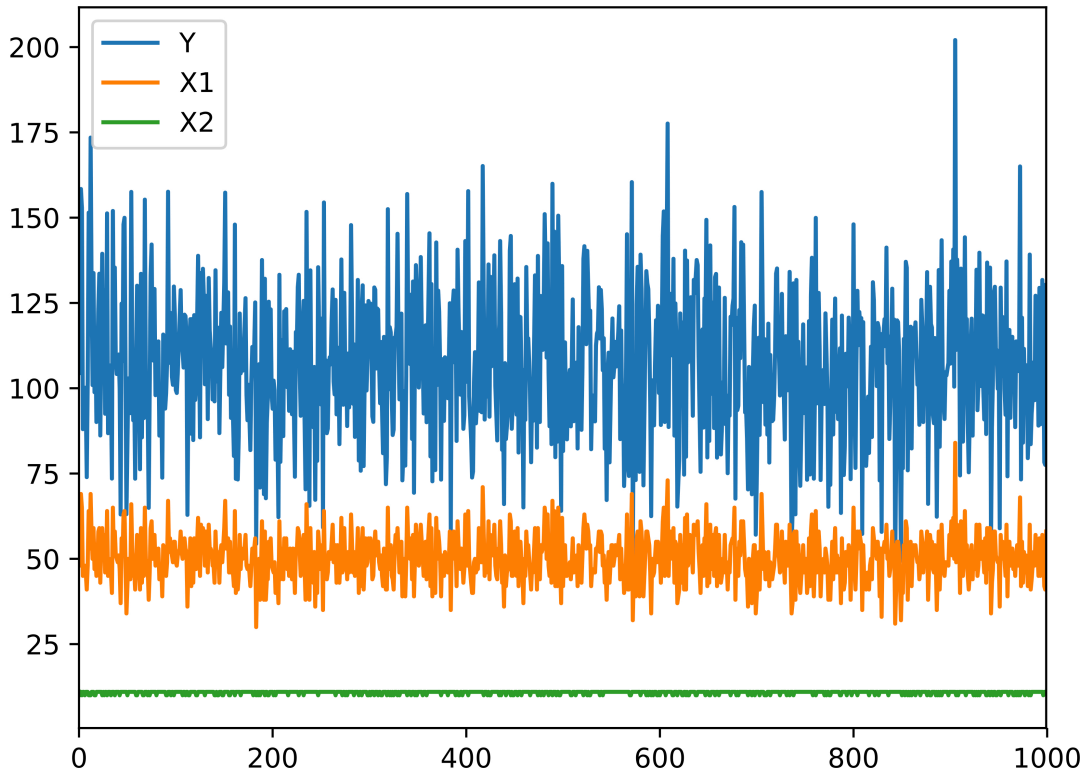
2.2.2 Create a plot in Python

File create_graph.py

```python
import pandas as pd
import matplotlib.pyplot as plt

sim_data = pd.read_csv("sample_data.csv")

plt.figure()
sim_data.plot()
plt.savefig("plot.pdf", format = "pdf")
```



2.2.3 Run statistical model in R

We can print the model output with:

```
sim_data <- read.csv("sample_data.csv")
summary(lm(Y ~ X1 + X2, data = sim_data))
```

```
##
## Call:
## lm(formula = Y ~ X1 + X2, data = sim_data)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -8.3988 -1.9452 -0.0261  2.0216  9.1066
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  9.09087    2.54667    3.57 0.000374 ***
## X1           3.00531    0.01326  226.68  < 2e-16 ***
## X2          -4.94658    0.22876  -21.62  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
```

```
## Residual standard error: 2.936 on 997 degrees of freedom
## Multiple R-squared:  0.9811, Adjusted R-squared:  0.981
## F-statistic: 2.585e+04 on 2 and 997 DF,  p-value: < 2.2e-16
```

2.2.4 Run statistical model in R

To save the output, we use the `sink` function.

File `estimate_model.R`

```
sim_data <- read.csv("sample_data.csv")
summary(lm(Y ~ X1 + X2, data = sim_data))

sink("estimation_summary.txt")
summary(lm(Y ~ X1 + X2, data = sim_data))
sink()
```

2.2.5 Makefile syntax

- `make` is a *command* that runs on a text file often named `Makefile`.
- A `Makefile` contains one or several blocks with the following structure:

```
targetfile: sourcefile(s)
[tab] command
```

2.2.6 Naive version

File: `Makefile`

```
sample_data.csv: simulate_data.R
    R CMD BATCH simulate_data.R

plot.pdf: create_graph.py
    python create_graph.py

estimation_summary.txt: estimate_model.R
    R CMD BATCH estimate_model.R
```

A simple call to `make` only builds the first target (`sample_data.csv`). To build the other targets, we have to use: `make plot.pdf` and `make estimation_summary.txt`.

2.2.7 Making all targets

File: `Makefile`

```
all: analysis

analysis: sample_data.csv plot.pdf estimation_summary.txt

sample_data.csv: simulate_data.R
    R CMD BATCH simulate_data.R

plot.pdf: create_graph.py
```

```
    python create_graph.py

estimation_summary.txt: estimate_model.R
    R CMD BATCH estimate_model.R
```

New data is simulated and saved in `sample_data.csv`. But `plot.pdf` and `estimation_summary.txt` are not updated.

2.2.8 Dealing with dependencies

- Problem `plot.pdf` and `estimation_summary.txt` depend on `sample_data.csv`.
- Solution: explicit dependencies.

File: `Makefile`

```
all: analysis

analysis: sample_data.csv plot.pdf estimation_summary.txt

sample_data.csv: simulate_data.R
    R CMD BATCH simulate_data.R

plot.pdf: sample_data.csv create_graph.py
    python create_graph.py

estimation_summary.txt: sample_data.csv estimate_model.R
    R CMD BATCH estimate_model.R
```

2.3 Git and GitHub

Guest lecture by Ista Zahn.

Chapter 3

Packages

3.1 Why?

- Organize your code
- Distribute your code
- Keep versions of your code

3.2 Package structure

- Folder hierarchy
  - `NAMESPACE`: package import / export
  - `DESCRIPTION`: metadata
  - `R/`: R code
  - `man/`: object documentation (with short examples)
  - `tests/`
  - `data/`
  - `src/`: compiled code
  - `vignettes/`: manual-like documentation
  - `inst/`: installed files
  - `demo/`: longer examples
  - `exec`, `po`, `tools`

3.3 Building steps

- `R CMD build`
- `R CMD INSTALL`
- `R CMD check`

3.3.1 `R CMD build`

```
R CMD build --help
```

Build R packages from package sources in the directories specified by 'pkgdirs'

3.3.2 R CMD INSTALL

```
R CMD INSTALL --help
```

Install the add-on packages specified by pkgs. The elements of pkgs can be relative or absolute paths to directories with the package sources, or to gzipped package 'tar' archives. The library tree to install to can be specified via '–library'. By default, packages are installed in the library tree rooted at the first directory in .libPaths() for an R session run in the current environment.

3.3.3 R CMD check

```
R CMD check --help
```

http://r-pkgs.had.co.nz/check.html

Check R packages from package sources, which can be directories or package 'tar' archives with extension '.tar.gz', '.tar.bz2', '.tar.xz' or '.tgz'.

A variety of diagnostic checks on directory structure, index and control files are performed. The package is installed into the log directory and production of the package PDF manual is tested. All examples and tests provided by the package are tested to see if they run successfully. By default code in the vignettes is tested, as is re-building the vignette PDFs.

3.3.4 Building steps with devtools

- `devtools::build`
- `devtools::install`
- `devtools::check`
- and many others: `load_all`, `document`, `test`, `run_examples`, ...

3.4 Create an R package

3.4.1 utils::package.skeleton

```
package.skeleton() # "in "fresh" session ("anRpackage")
package.skeleton("pkgname") # in "fresh" session

set.seed(02138)
f <- function(x, y) x+y
g <- function(x, y) x-y
d <- data.frame(a = 1, b = 2)
e <- rnorm(1000)
package.skeleton(list = c("f","g","d","e"), name = "pkgname")
```

3.4.2 devtools::create

```
devtools::create("path/to/package/pkgname")
```

3.4.3 Submit to CRAN

http://r-pkgs.had.co.nz/release.html

3.5 R packages on GitHub

http://r-pkgs.had.co.nz/git.html

3.5.0.1 RStudio and GitHub integration

Command line

```
git init
git add *
git commit -m "First commit"
git remote add origin git@github.com:harvard-P01/pkgtemplate.git
git push -u origin master
```

3.5.1 Installing from GitHub

```
devtools::install_github("harvard-P01/pkgtemplate")
```

```
devtools::install_github("harvard-P01/pkgtemplate",
                         build_vignettes = TRUE)
```

3.5.2 .gitignore

RStudio default

```
.Rproj.user
.Rhistory
.RData
```

GitHub default

```
# History files
.Rhistory
.Rapp.history

# Example code in package build process
*-Ex.R

# RStudio files
.Rproj.user/
```

```
# produced vignettes
vignettes/*.html
vignettes/*.pdf
```

3.6 RStudio projects

- `.Rproj` file extension, in our example `pkgtemplate.Rproj`
- A project has its own:
  - R session
  - .Rprofile (*e.g.*, to customize startup environment)
  - .Rhistory
- Default working directory is project directory
- Keeps track of project-specific recent files

3.6.1 Project options

```
Version: 1.0

RestoreWorkspace: Default
SaveWorkspace: Default
AlwaysSaveHistory: Default

EnableCodeIndexing: Yes
UseSpacesForTab: Yes
NumSpacesForTab: 2
Encoding: UTF-8

RnwWeave: knitr
LaTeX: pdfLaTeX

AutoAppendNewline: Yes
StripTrailingWhitespace: Yes

BuildType: Package
PackageUseDevtools: Yes
PackageInstallArgs: --no-multiarch --with-keep.source
```

3.6.2 Package documentation

- Functions and methods
- Vignettes
  - PDF
  - `knitr` (or `Sweave`)

3.7 Package workflow example

Creating R Packages: A Tutorial (Friedrich Leisch, 2009)

- https://cran.r-project.org/doc/contrib/Leisch-CreatingPackages.pdf

3.7.1 Add `linreg.R` to `R/` directory

```
linmodEst <- function(x, y) {
  ## compute QR-decomposition of x
  qx <- qr(x)
  ## compute (x'x)^(-1) x'y
  coef <- solve.qr(qx, y)
  ## degrees of freedom and standard deviation of residuals
  df <- nrow(x) - ncol(x)
  sigma2 <- sum((y - x %*% coef) ^ 2) / df
  ## compute sigma^2 * (x'x)^-1
  vcov <- sigma2 * chol2inv(qx$qr)
  colnames(vcov) <- rownames(vcov) <- colnames(x)
  list(
    coefficients = coef,
    vcov = vcov,
    sigma = sqrt(sigma2),
    df = df
  )
}
```

3.7.2 Run our function

```
data(cats, package = "MASS")
linmodEst(cbind(1, cats$Bwt), cats$Hwt)
```

```
## $coefficients
## [1] -0.3566624  4.0340627
##
## $vcov
##             [,1]        [,2]
## [1,]  0.4792475 -0.17058197
## [2,] -0.1705820  0.06263081
##
## $sigma
## [1] 1.452373
##
## $df
## [1] 142
```

We can compare the output with `lm`.

```
lm1 <- lm(Hwt ~ Bwt, data=cats)
lm1
```

```
##
## Call:
```

```
## lm(formula = Hwt ~ Bwt, data = cats)
##
## Coefficients:
## (Intercept)          Bwt
##     -0.3567       4.0341
```

```
coef(lm1)
```

```
## (Intercept)         Bwt
##  -0.3566624   4.0340627
```

```
vcov(lm1)
```

```
##             (Intercept)         Bwt
## (Intercept)   0.4792475 -0.17058197
## Bwt          -0.1705820  0.06263081
```

```
summary(lm1)$sigma
```

```
## [1] 1.452373
```

3.7.3 Add ROxygen2 documentation

```
#' Linear regression
#'
#' Runs an OLS regression not unlike \code{\link{lm}}
#'
#' @param y response vector (1 x n)
#' @param X covariate matrix (p x n) with no intercept
#'
#' @return A list with 4 elements: coefficients, vcov, sigma, df
#'
#' @examples
#' data(mtcars)
#' X <- as.matrix(mtcars[, c("cyl", "disp", "hp")])
#' y <- mtcars[, "mpg"]
#' linreg(y, X)
#'
#' @export
#'
linmodEst <- function(x, y) {
  ## compute QR-decomposition of x
  qx <- qr(x)
  ## compute (x'x)^(-1) x'y
  coef <- solve.qr(qx, y)
  ## degrees of freedom and standard deviation of residuals
  df <- nrow(x) - ncol(x)
  sigma2 <- sum((y - x %*% coef) ^ 2) / df
  ## compute sigma^2 * (x'x)^-1
  vcov <- sigma2 * chol2inv(qx$qr)
  colnames(vcov) <- rownames(vcov) <- colnames(x)
  list(
    coefficients = coef,
    vcov = vcov,
    sigma = sqrt(sigma2),
```

```
    df = df
  )
}
```

3.7.4 Configure Build Tools

3.7.5 man/linmodEst.Rd

```
% Generated by roxygen2 (4.1.1): do not edit by hand
% Please edit documentation in R/linmodEst.R
\name{linmodEst}
\alias{linmodEst}
\title{Linear regression}
\usage{
linmodEst(x, y)
}
\arguments{
\item{y}{response vector (1 x n)}

\item{X}{covariate matrix (p x n) with no intercept}
}
\value{
A list with 4 elements: coefficients, vcov, sigma, df
}
\description{
Runs an OLS regression not unlike \code{\link{lm}}
}
\examples{
data(mtcars)
X <- as.matrix(mtcars[, c("cyl", "disp", "hp")])
y <- mtcars[, "mpg"]
linmodEst(y, X)
}
```

3.7.6 Formatted output

3.7.7 DESCRIPTION

```
Package: pkgtemplate
Type: Package
Title: What the Package Does (Title Case)
Version: 0.1
Date: 2015-10-24
Author: Who wrote it
Maintainer: Who to complain to <yourfault@somewhere.net>
Description: More about what it does (maybe more than one line)
License: What license is it under?
LazyData: TRUE
```

3.7.8 NAMESPACE

export's automatically generated when parsing ROxygen2 snippets

```
export(linmodEst)
```

3.7.9 S3 basics

```
hello <- function() {
 s <- "Hello World!"
 class(s) <- "hi"
 return(s)
}

hello()
```

```
## [1] "Hello World!"
## attr(,"class")
## [1] "hi"
```

3.7.10 S3 basics

```
print.hi <- function(...) {
  print("Surprise!")
}

hello()
```

```
## [1] "Surprise!"
```

3.7.11 S3 and S4 generics

```
linmod <- function(x, ...)
  UseMethod("linmod")
```

```
linmod.default <- function(x, y, ...) {
  x <- as.matrix(x)
  y <- as.numeric(y)
  est <- linmodEst(x, y)
  est$fitted.values <- as.vector(x %*% est$coefficients)
  est$residuals <- y - est$fitted.values
  est$call <- match.call()
  class(est) <- "linmod"
  return(est)
}
```

3.7.12 print

```
print.linmod <- function(x, ...) {
  cat("Call:\n")
  print(x$call)
  cat("\nCoefficients:\n")
  print(x$coefficients)
}
```

```
x <- cbind(Const = 1, Bwt = cats$Bwt)
y <- cats$Hw
mod1 <- linmod(x, y)
mod1
```

```
## Call:
## linmod.default(x = x, y = y)
##
## Coefficients:
##      Const        Bwt
## -0.3566624  4.0340627
```

3.7.13 Other methods

- summary.linmod
- print.summary.linmod
- predict.linmod
- plot.linmod
- coef.linmod, vcov.linmod, ...

3.7.14 Formulas and model frames

```
linmod.formula <- function(formula, data = list(), ...) {
  mf <- model.frame(formula = formula, data = data)
  x <- model.matrix(attr(mf, "terms"), data = mf)
  y <- model.response(mf)
  est <- linmod.default(x, y, ...)
  est$call <- match.call()
  est$formula <- formula
  return(est)
}
```

3.8 Unit testing

3.8.1 Unit tests and testthat

http://r-pkgs.had.co.nz/tests.html

In package directory:

```
devtools::use_testthat()
```

pre-populates `test/testthat/`

Test files should start with `test` to be processed.

3.8.2 `test_coef.R`

```
data(cats, package = "MASS")
l1 <- linmod(Hwt ~ Bwt * Sex, data = cats)
l2 <- lm(Hwt ~ Bwt * Sex, data = cats)

test_that("same estimated coefficients as lm function", {
  expect_equal(l1$coefficients, l2$coefficients)
})
```

```
==> devtools::test()

Loading pkgtemplate
Loading required package: testthat
Testing pkgtemplate
.
Woot!
```

3.9 Code coverage

3.10 Continuous integration

3.11 Vignettes

http://r-pkgs.had.co.nz/vignettes.html

```
devtools::use_vignette("linmod")
```

https://github.com/harvard-P01/pkgtemplate/blob/master/vignettes/linmod.Rmd

Chapter 4

Optimization

In this Chapter, we will see how to measure and improve code performance.

4.1 Measuring performance

4.1.1 Profiling

4.1.2 Benchmarking

4.2 Improving performance

4.2.1 Introduction to C/C++

4.2.2 Rcpp

Chapter 5

Databases

5.1 Overview

5.2 SQL

5.3 noSQL

5.4 R interfaces

Chapter 6

Big data

In this Chapter, we are going to review different approaches to handle and perform analyses on *data that does not fit in memory*.

6.1 Reading big data (that fits in memory)

6.1.1 R package comparison

6.1.2 Python

6.2 Sampling (can be read, not analyzed easily)

6.3 Pure R solutions

6.4 JVM solutions

6.4.1 h20

6.4.2 Spark

Chapter 7

Visualization

7.1 Principles of visualization

7.2 Maps and GIS

Bibliography

Blackwell, M. and Sen, M. (2012). Large datasets and you: A field guide. *The Political Methodologist*, 20(1):2–5.

Harper, F. M. and Konstan, J. A. (2015). The movielens datasets: History and context. *ACM Trans. Interact. Intell. Syst.*, 5(4):19:1–19:19.