Applied Statistics: R

Semester 2018-I

Francisco Rosales Marticorena

Academic Department of Finance

Table of Contents:

1 Introduction

2 Conditioning and Condition Numbers

3 Numerics

4 Simulations

5 R Basics

Introduction

Objectives

- new graphical methods
- data mangement (import/export)
- functions, debugging, condition handling
- object-oriented programing: S3 and S4 classes
- parallelization of R code
- integration of other programing languages, e.g. C++
- building R packages

Coordination & Examination

- No Blackboard.
- For lecture notes, code, references, etc. go here: `https://github.com/franciscorosales-marticorena/ApplStatsR`
- One exam on mid-term week. 20% of your grade.

Reading Material

- Hadley Wickham. Advanced R. The R series. CRC Press, 2015. Available online: `http://adv-r.had.co.nz/`.
- Owen Jones, Robert Maillardet, and Andrew Robinson. Introduction to scientific programming and simulation using R. Chapman & Hall/CRC, 2009.
- Brian Everitt and Torsten Hothorn. A handbook of statistical analyses using R. Chapman & Hall/CRC, 2006.
- R Data Import/Export manual. Available online: `http://cran.r-project.org/doc/manuals/r-release/R-data.pdf`
- Deepayan Sarkar. Lattice: multivariate data visualization with R. Use R! Springer, 2008.
- Roger S Bivand, Edzer J Pebesma, and Virgilio Gómez-Rubio. Applied spatial data analysis with R, volume 10 of Use R! Springer, 2008.

Who are you?

- operating system: Linux, Apple, Windows?
- text editor for R: Rstudio, emacs, Vim?
- other programing languages: C, C++, Java, Python, Fortran, Julia?
- compiled already an R package?
- LaTeX?

Conditioning and Condition Numbers

Two fundamental problems in numerical analysis:

1. **Conditioning:** pertains to the perturbation behaviour of a math problem.
2. **Stability:** pertains to the perturbation behaviour of an algorithm used to solve a math problem.

Condition of a Problem

Definition 2.1 (Problem)
Let $f : X \to Y$ denote a problem from a normed vector space of data to a normed vector space of solutions. Usually we assume f continuous and possibly non-linear.

Definition 2.2 (Problem instance)
We call the combination of a problem f with available data x as a problem instance.

1. *well-conditioned problem instance: a small perturbation in x leads to small changes in $f(x)$.*
2. *ill-conditioned problem instance: a small perturbation in x leads to large changes in $f(x)$.*

Absolute Condition Number

Definition 2.3 (Absolute condition number)

The absolute condition number of f at x is defined as

$$\hat{\kappa} := \sup_{\delta x} \frac{\|\delta f\|}{\|\delta x\|},$$

where δx is an infinitesimally small perturbation in x, and $\delta f := f(x + \delta x) - f(x)$.

Remark 2.1
Note that if f is differentiable, then $\lim_{\delta x \to 0} \delta f = J(x)\delta x$, where $J(x)$ is the Jacobian of f at x. Hence, the condition number is simply

$$\hat{\kappa} = \sup_{\delta x} \frac{\|J(x)\delta x\|}{\|\delta x\|} = \|J(x)\|,$$

where $\|J(x)\|$ is the norm of $J(x)$ induced by X and Y.

Relative Condition Number

Definition 2.4 (Relative Condition Number)
The relative condition number is defined as

$$\kappa(x) = \sup_{\delta x} \left(\frac{\|\delta f\| / \|f(x)\|}{\|\delta x\| / \|x\|} \right).$$

Once again, if f is differentiable,

$$\kappa(x) = \frac{\|J(x)\|}{\|f(x)\| / \|x\|}.$$

Remark 2.2
A problem is well-conditioned if $\kappa(x)$ is small, e.g. $1, 10, 10^2$ and ill-conditioned if $\kappa(x)$ is large, e.g. $10^6, 10^{16}$.

Example 2.1
Consider the problem of obtaining the scalar $x/2$ from $x \in \mathbb{C}$.

Lösung.
Let $f : x \to x/2$. The Jacobian reads $J(x) = 1/2$, and

$$\kappa(x) = \frac{\|J(x)\|}{\|f(x)\|/\|x\|} = \frac{1/2}{(\|x\|/2)/\|x\|} = 1,$$

thus the problem is well-conditioned. ∎

Example 2.2
Consider the problem of obtaining $\sqrt{x}$ from $x \in \mathbb{R}^+$.

Lösung.
Let $f : x \to \sqrt{x}$. The Jacobian reads $J(x) = 1/(2\sqrt{x})$, and

$$\kappa(x) = \frac{\|J(x)\|}{\|f(x)\|/\|x\|} = \frac{1/(2\sqrt{x})}{\sqrt{x}/x} = \frac{1}{2},$$

thus the problem is well-conditioned as well. ∎

Example 2.3
Consider the problem of obtaining the scalar $f(\boldsymbol{x}) = x_1 - x_2$ *from* $\boldsymbol{x} \in \mathbb{C}^2$*. Use the* L_∞ *norm.*

Lösung.
The Jacobian is then

$$\boldsymbol{J}(\boldsymbol{x}) = \begin{bmatrix} \frac{\partial f}{\partial x_1} & \frac{\partial f}{\partial x_2} \end{bmatrix} = \begin{bmatrix} 1 & -1 \end{bmatrix}, \quad \|\boldsymbol{J}(\boldsymbol{x})\|_\infty = 2.$$

and the relative condition number is

$$\kappa(x) = \frac{\|\boldsymbol{J}(\boldsymbol{x})\|_\infty}{\|f(\boldsymbol{x})\|/\|\boldsymbol{x}\|} = \frac{2}{|x_1 - x_2|/\max\{|x_1|, |x_2|\}},$$

hence as $|x_1 - x_2| \to 0$, $\kappa(x) \to \infty$ and the problem is ill-conditioned. ■

Example 2.4

Let $f(x) = ax^2 + bx + c$. Consider the problem of computing the roots of $f(x)$, given $a = 1$, $b = -2$ and $c = 1$, and illustrate the sensitivity of the solution if $c' = 1 + \Delta$, where $\Delta = -10^{-4}$.

Lösung.

$$
\begin{aligned}
f(x) &= x^2 - 2x + 1 = (x-1)^2, \quad x = 1, \\
f'(x) &= x^2 - 2x + 0.9999 = (x-0.99)(x-1.01), \quad x_1 = 0.99, x_2 = 1.01
\end{aligned}
$$

leading to

$$
\kappa(x) = \sup_{\delta x} \left(\frac{\|\delta f\| / \|f(x)\|}{\|\delta x\| / \|x\|} \right) = \sup_{\delta x} \left(\frac{10^{-4}/0}{10^{-4}/1} \right) = \frac{1}{0} = \infty.
$$

■

Numerics

Basics

Representation of numbers

- Computers use indicators to encode information (1 for ON, 0 for OFF)
- One indicator is called a bit, eight bits are one byte, . . .
- The way how numbers are represented depends on your computer, R has no influence on that

Representation of integers

The sign-and-magnitude scheme

- Use one bit for the sign ($\pm$) and the rest for the magnitude
- For k bits an integer is represented as

$$\pm b_{k-2} \ldots b_2 b_1 b_0,$$

where each b_i is 0 or 1 and which is translated to

$$\pm(2^0 b_0 + 2^1 b_1 + \cdots + 2^{k-2} b_{k-2})$$

Example 3.1
For $k = 8$, -1001101 represents the number

$$-(2^0 \cdot 1 + 2^1 \cdot 0 + 2^2 \cdot 1 + 2^3 \cdot 1 + 2^4 \cdot 0 + 2^5 \cdot 0 + 2^6 \cdot 1) = -77.$$

Representation of integers

Properties and extensions

- Integers range symmetrically from $-(2^{k-1}-1)$ to $2^{k-1}-1$
- Two representations of 0 ($\pm$)
- The biased scheme1 avoids -0, but addition of integers is complex and slow
- The two's complement scheme uses the binary representation for positive integers and represents $-1, -2, \ldots, -2^{k-1} via 2^k - 1, 2^k - 2, ..., 2^k - 2^{k-1}$
- Efficient implementation of addition (equiv to addition modulo 2^k)

in R

```
> .Machine$integer.max
[1] 2147483647
```

Representation of real numbers

Properties:

Computers need to limit the size of the mantissa and exponent. In double precision

- use 8 bytes (i.e. 64 bits) in total
- 1 bit for the sign
- 52 bits for the mantissa
- 11 bits for the exponent (representation via biased scheme, ranges from -1022 to 1024)

in R

```
> .Machine$double.xmin
[1] 2.225074e-308
> .Machine$double.xmax
[1] 1.797693e+308
```

Representation of real numbers

Further examples:

- Underflow/ overflow

```
> 2^1023 + 2^1022 + 2^1021
[1] 1.572981e+308
> 2^1023 + 2^1022 + 2^1022
[1] Inf
```

- "Asymmetry"

```
> 2^(-1074) == 0
[1] FALSE
> 2^(-1075) == 0
[1] TRUE
> 1 / 2^(-1074)
[1] Inf
```

Representation of real numbers

- Machine epsilon (smallest x, s.t. $1 + x$ can be distinguished from 1); round off

```
> x <- 1 + 2 ^ -52
> x - 1 == 0
[1] FALSE
> y <- 1 + 2 ^ -53
> y - 1 == 0
[1] TRUE
```

Representation of real numbers

Numerical errors:

Numerical errors occur all the time. E.g. there is no finite binary representation of 0.1. Denote by $\tilde{x}$ an approximation of x.

- The absolute error is defined as $|x - \tilde{x}|$
- The relative error is defined as $\frac{|x-\tilde{x}|}{x}$

Catastrophic cancellation describes a loss of accuracy with a relative error of 10^{-8} or larger due to error propagation. It can occur when subtracting numbers of similar size.

Example

Computation of $\sin(x) - x$ to 0

- Standard computation

$$\lim_{x \to 0} \frac{\sin(x)}{x} = 1 \Rightarrow \sin(x) \approx x \quad \text{near} \quad 0$$

- Taylor expansion of order 2

$$\sin(x) - x \approx -\frac{x^3}{6} + \frac{x^5}{120} = -\frac{x^3}{6}\left(1 - \frac{x^2}{20}\right)$$

```
> x = 2^-c(10, 20, 30)
> sin(x) - x
[1] -1.552204e-10 -1.445250e-19  0.000000e+00
> -x^3 / 6 * (1 - x^2 / 20)
[1] -1.552204e-10 -1.445603e-19 -1.346323e-28
```

- Relative errors: $\approx 10^{-11}$, 10^{-4}, 1
- Catastrophic cancellation at $x = 2^{-20}$ and $x = 2^{-30}$!

Numerical Algorithms

Generalities

- For many mathematical and statistical problems there are no analytical solutions (or they are very hard to find)
- Examples are optimization, integration, solving (systems of) equations, differentiation, eigenvalue problems, . . .
- For most cases, numerical alternatives have been developed and implemented
- Key features of such algorithms are accuracy/precision and convergence (rate)/speed

Numerical Algorithms

Some important functions for numerical mathematics in R:

optimization	derivation	(system of) equations	integration	other
`optim(ize)`	`deriv`	`solve`	`integrate`	`eigen`
`optimx`	`grad`	`polyroot`	`adaptIntegrate`	`qr`
`nlm`	`hessian`	`solveLP`		

See `http://cran.r-project.org/web/views/Optimization.html` for more.

Note that also other functions like `glm` or `lme` use numerical techniques to optimize the likelihood with respect to the regression parameters.

Example

Find the minimum of the function $f(x) = x^2$. What is the value of $\int_{-2}^{2} x^2 dx$?

```
> my.square = function(x) {
+ x^2
+}
> optimize(f = my.square, interval = c(-2, 2)) $minimum
[1] -5.551115e-17
$objective
[1] 3.081488e-33
> optimize(f = my.square, interval = c(2, 3))
$minimum
[1] 2.000066
$objective
[1] 4.000264
> integrate(f = my.square, lower = -2, upper = 2)
5.333333 with absolute error < 5.9e-14
```

Floating Point Arithmetic

- A computer uses a finite number of bits to represent $x \in \mathbb{R}$.
- In fact, only $x \in \mathcal{S} \subset \mathbb{R}$ can be represented by a computer.

Remark 3.1
The IEEE double precision arithmetic follows these follows these rules wrt any number $x \in \mathbb{R}$:

1. $x < 1.79 \times 10^{308}$
2. $x > 2.23 \times 10^{-308}$
3. $\Delta x = 2^{-52}$

Remark 3.2
The problem of not being able to represent a very large (very small) number is called overflow (underflow).

Example 3.2

1. *Print all $x \in [1, 2]$ using the IEEE double precision arithmetic.*

$$1, 1 + 1 \times 2^{-52}, 1 + 2 \times 2^{-52}, 1 + 3 \times 2^{-52}, \ldots, 2$$

2. *Print all $x \in [2^j, 2^{j+1}]$ using the IEEE double precision arithmetic.*

$$2^j, 2^j + 1 \times 2^{-52+j}, 2^j + 2 \times 2^{-52+j}, 2^j + 3 \times 2^{-52+j}, \ldots, 2^{j+1}$$

Remark 3.3

Note that the number of reals between two consecutive integers in the same regardless of value of the integers.

Floating Point Number System (FPNS)

Characteristics:

- the position of the decimal (or binary) point is stored separate from the digits.
- the gaps between adjacent represented numbers scale in proportion to the size of the numbers.

Example 3.3 (Idealized FPNS)
Let $\mathbb{F} \subset \mathbb{R}$ denote a discrete set, formed by 0 together with all the numbers of the form

$$x = \pm(m/\beta^t)\beta^e,$$

where $\beta \geq 2$ is the base, $t \geq 1$ is the precision (24 and 53 for IEEE single and double precision resp.), $e \in \mathbb{Z}$ is the exponent and $1 \leq m \leq \beta^t$.

Machine Epsilon

Definition 3.1 (Machine Epsilon)
A working definition of Machine epsilon is, simply, half the distance between 1 *and the next floating point number, i.e.*

$$\epsilon_{machine} = \frac{1}{2}\beta^{1-t}.$$

For the IEEE double precision we obtain $\epsilon_{machine} = 2^{-53}$*. Intuitively, it provides an idea of the FPNS's resolution.*

Proposition 3.1
The following two assertions are equivalent

1. *For all* $x \in \mathbb{R}$, $\exists x' \in \mathbb{F} : |x - x'| \leq \epsilon_{machine}|x|$.
2. *For all* $x \in \mathbb{R}$, $\exists \epsilon : |\epsilon| \leq \epsilon_{machine}$, $fl(x) = x(1+\epsilon)$,

where $fl : \mathbb{R} \to \mathbb{F}$ *gives the closest floating point rep. to a real number.*

- $+$, $-$, $\times$, $\div$ are operations in $\mathbb{R}$.
- $\oplus$, $\ominus$, $\otimes$, $\oslash$ are the corresp. analogues in $\mathbb{F}$.

Proposition 3.2
Let $x, y \in \mathbb{F}$ and let $*$ denote any operation $+$, $-$, $\times$, $\div$, and $\circledast$ its floating point analogue. If a computer system is such that

$$x \circledast y = fl(x * y),$$

then for all $x, y \in \mathbb{F}$, $\exists \epsilon : |\epsilon| \leq \epsilon_{machine}$, $x \circledast y = (x * y)(1 + \epsilon)$.

Simulations

Generalities

Definition 4.1
A (Monte-Carlo)-Simulation is a numerical technique for conducting experiments on a computer. The term Monte-Carlo refers to the involvement of random experiments.

Application areas:
Simulation studies are performed when analytical results are hard or impossible to find to

- identify properties of estimators or test statistics (bias, variance, distribution, etc.)
- investigate consequences of violations of model assumptions
- find out about the influence of the sample size
- compare different models or estimators (in terms of bias, precision, computational time, etc.)

Generalities

Rationale

- Estimators and test statistics have true sampling distributions (under certain assumptions)
- Knowing the distribution would answer all questions about the properties described above
- Approximate these distributions by conducting according random experiments very often (law of large numbers)

Usual setup

- Simulate K independent data sets under the conditions of interest
- Calculate the numerical values of the statistic T of interest for each data set, i.e. $T_1, \ldots, T_K$
- Evaluate the properties of the results under the assumption that the distribution of $T_1, \ldots, T_K$ approximates the true distribution of the statistic

Stochastic distributions in R

In R, density (***d***), distribution (***p***), quantile (***q***), and (pseudo) random number generator (***r***) functions are already implemented.

function	distribution
`beta()`	beta-
`binom()`	binomial-
`exp()`	exponential-
`gamma()`	gamma-
`hyper()`	hypergeometric-
`logis()`	logistic-
`lnorm()`	lognormal-
`nbinom()`	negativ binomial-
`norm()`	normal-
`pois()`	poisson-
`t()`	t-
`unif()`	uniform-

Stochastic distributions in R

Further notes

- The random numbers in R are not really random
- Use set.seed to make your results replicable

```
> set.seed(123)
> rnorm(3)
[1] -0.5604756 -0.2301775  1.5587083
> rnorm(3)
[1] 0.07050839 0.12928774 1.71506499
> set.seed(123)
> rnorm(3)
[1] -0.5604756 -0.2301775  1.5587083
> set.seed(123)
> rnorm(3)
[1] -0.5604756 -0.2301775  1.5587083
```

Case study

Stochastic Frontier type data:
Stochastic Frontier Analysis (SFA) belongs to the field of productivity analysis

- Aim: quantify inefficiency and determine a production function
- Assumptions: deviations from the production function (the errors) are a combination of stochastic noise and inefficiency, formally $\epsilon = v - u$.
- Comparison: the model formulation deviates from the classical linear model only in terms of inefficiency

Case Study

Investigate the behavior of the estimator for the linear regression model without intercept

$$y_i = \beta x_i + \epsilon_i, \quad i = 1, ..., n, \tag{1}$$

when the distributional assumption $\epsilon_i \overset{iid}{\sim} \mathcal{N}(0, \sigma^2)$ is violated.

More precisely, simulate $K = 50$ independent data sets of sample size $n = 200$ with

- $x_i \sim \mathcal{N}(3, 2)$
- $\epsilon_i = u_i - v_i$, where $u_i \sim \mathcal{N}(0, 1^2)$ and $v_i \sim \mathcal{N}_+(0, 1^2)$
- $\beta = 2$
- y_i according to (1).

and estimate the covariate effect for each of the data sets. What are the approximate mean and variance of $\hat{\beta}$?

Further Reading

- Jones, Maillardet and Robinson (2009): Scientific Programming and Simulation Using R
- `http://cran.r-project.org/web/views/NumericalMathematics.html`
- `http://cran.r-project.org/web/views/Optimization.html`
- `http://cran.r-project.org/web/views/Distributions.html`

R Basics

Really Basic

help:	`?topic, help(topic), args(some function)`
assignments:	`x <- 5 (recommended), x = 5, 5 -> x`
operators:	`+, -, *, /, ^, &, &&, |, ||; see help("+")`
comparisons:	`==,! =,>,>=,<,<=;see help("= = ")`
loops:	`for, while, repeat`
comments:	`everything that follows #`

case sensitive: usage of CAPITAL and small letters matters!

Basic Data Structures

integer:	`1,2,301L`
double:	`1.0, .141, 1.23e-3, NaN, Inf, -Inf`
logical:	`TRUE, FALSE`
character:	`"hello", "I'm a string"`
numeric	`general class for 'numberliness'` (integer, double)
complex	`1+0i, 1i, 3+5i`
missings:	`NA`

```
mode(1:10)           ## [1] "numeric"
storage.mode(1:10)   ## [1] "integer"
mode(pi)             ## [1] "numeric"
storage.mode(pi)     ## [1] "double"
mode(TRUE)           ## [1] "logical"
mode("Hello")        ## [1] "character"
### also useful:
str(1:10)            ## int [1:10] 1 2 3 4 5 6 7 8 9 10
str(LETTERS)         ## chr [1:26] "A" "B" "C" "D" "E" ...
```

Construction & Coercion

- the constructor for a data type is named like the data type
- vector can be constructed with `c()`
- coerce to a type `xxx` by as `.xxx()`
- when combining different data types, they will be coerced to the most flexible type
- coercion often happens automatically
- check if `xxx` is a specific type by is `.xxx()`

```
integer(5)            ## [1] 0 0 0 0 0
double(0)             numeric(0)
str(c("a",1))         ##chr [1:2] "a" "1"
x <- c(FALSE, FALSE, TRUE)
as.numeric(x)         ## [1] 0 0 1
# Total number of TRUEs
sum(x)                ## [1] 1
```

NA, NaN and NULL

In R, there three ways to represent "nothing", but the reason for the missingness of the information can be distinguished:

NA	missing sample values, impossible coercion, . . .
NaN	undefined results in mathematical operation (e.g. `log(-1)`, `1/0`)
NULL	null pointer, i.e. pointer in empty/undefined memory

```
c(3, NA)    ## [1] 3 NA
c(3, 0/0)   ## [1] 3 NaN
c(3, NULL)  ## [1] 3
max(3, NA)  ## [1] NA
```

Inifinity

Some mathematical operations can be performed with `Inf` and `-Inf`:

```
max(3, Inf)
## [1] Inf
```

```
min(3, Inf)
## [1] 3
```

```
c(Inf + Inf, (-Inf) * Inf, Inf - Inf)
## [1] Inf -Inf NaN
```

Complex Data Types

- complex data structures in R can be organized by their dimensionality and if all their contents are of the same type, or not:
- a data.frame is a matrix, in which not every column has to have the same data type, and a list, in which each element has the same length

```
x <- matrix(1, nrow = 5, ncol = 2)
is.matrix(x)                        ## [1] TRUE
as.vector(x)                        ## [1] 1 1 1 1 1 1 1 1 1 1
x <- list(a = "Hallo", b = 1:10, pi = pi)
```

Attributes

All objects can have arbitrary additional attributes, used to store metadata about the object.

- can be thought of as a named list (with unique names); other frequently encountered attributes: "dimnames", "names", "class"(!)
- can be accessed individually with attr() or all at once (as a list) with attributes().
- arrays are simply vectors with a "dim"-attribute.
- factor is a vector with attribute levels
- as.xxx() functions delete all attributes including dimensionality

Attributes

Example 5.1

```
x <- matrix(1:10, ncol = 5)
attributes(x)               ## $dim
                            ## [1] 2 5
rownames(x) <- c("Eins", "Zwei")
attributes(x)               ## $dim
                            ## [1] 2 5
                            ## $dimnames
                            ## $dimnames[[1]]
                            ## [1] "Eins" "Zwei"
                            ## $dimnames[[2]]
                            ## NULL
as.character(x)             ## [1] "1"  "2"  "3"  "4"  "5"  "6"...
attributes(as.character(x))  ## NULL
```

Subsetting

- There are three subsetting operators: `[`, `[[`, `and $`
- the three types of subsetting:
  - Positive integers return elements at the specified positions.
  - Logical vectors select elements where the corresponding logical value is TRUE; application of logical expressions.
  - character vectors to return elements with matching names.
- important differences in behavior of different objects (e.g., vectors, lists, factors, matrices, and data frames).
- More advanced subsetting, in particular in combination with complex logical expressions, can be done using the functions subset() and which().
- There are also two additional subsetting operators that are needed for S4 objects: `@` (equivalent to `$`), and `slot()` (equivalent to `[[`).
- The default drop=TRUE simplifies the data type of the result.

Atomic Vectors

Use "["-operator and number, logical vector or name of the element you want to pull out.

```
x <- c(2.1, 4.2, 3.3, 5.4)
x[c(3, 1)]                      ## [1] 3.3 2.1
x[-c(3, 1)]                     ## [1] 4.2 5.4
x[c(TRUE, TRUE, FALSE, FALSE)]  ## [1] 2.1 4.2

(y <- setNames(x, letters[1:4])) ##   a   b   c   d
                                 ## 2.1 4.2 3.3 5.4
y[c("d", "c", "a")]              ##  d   c   a
                                 ## 5.4 3.3 2.1
```

Matrices & Arrays

Subsetting matrices and arrays with “[”-operator like vectors, while the dimension is separated by comma:

```
x <- matrix(1:10, ncol=2); colnames(x) = c("Eins", "Zwei")
x[1:2,]                   # results a matrix
x[,"Zwei"]                # results a vector
x[,"Zwei", drop=FALSE] # results a matrix
x[-3,]                    # everything, but not third row
```

Lists

Subsetting lists with “[”-operator returns always a list, while [[, and $ pull out elements of the list:

```
(x <- list(a = "Hallo", b = 1:10, pi = pi))
x$a       # first element of the list
x[['a']]
x[[1]]
x[1]      # list with one element
x[2:3]    # list with two elements
x[[2:3]]  # wrong result
```

Data Frames

Data frames possess the characteristics of both lists and matrices: if you subset with a single vector, they behave like lists; if you subset with two vectors, they behave like matrices.

```
iris[1:10,]            # data frame with 10 rows
iris[,1]               # numerical
iris$Sepal.Length      # the name
iris$Sepa1.Length      #Oops! what happened?
iris[,"Sepal.Length"]  # again first column
iris[,"Sepa1.Length"]  # Error:  undefined columns selected
iris[,1, drop=FALSE]   # data frame with one column
```

Flow Control

- conditional evaluation: if, else, ifelse
- loops: for, while, repeat, switch

basic vocabulary:

```
if, &&, || (short circuiting)
for, while, repeat
next, break
switch
ifelse
```

if/else Conditions

```
if (<test>) {
  <expression1>
} else {
  <expression2>
}
```

- `else` block is optional
- `<test>` has to result in a value that can be converted to a logical value
- only the first element of `<test>` is used, otherwise a warning is triggered
- for the evaluation of more than one statement use `&`, `|` or `all()` and `any()`
- can be nested

for Loop

In each interation, `<var>` is set to the next element of `<vector>` and `<expression>` is evaluated.

```
for (<var> in <vector>) {<expression>}
sum <- 0
for(i in 1 : length(x)) {
sum <- sum + x[i]
}
sum <- 0
for(x_value in x) {   ## more efficient
sum <- sum + x_value
}
```

Use `seq_along(x)` instead of `1:length(x)`:

```
x <- numeric(0)
1 : length(x)       ## [1] 1 0
seq_along(x)        ## integer(0)
```

`while` Loop

As long `<test>` is TRUE the `<expression>` is evaluated.

```
while(<test>) {
  <expression>
}
```

E.g., the sum until the first `NA`:

```
sum <- 0
i <- 1
while((i <= length(x)) && !is.na(x[i])) {
  sum <- sum + x[i]
i <- i + 1 }
```

Be aware of infinite loops!

next & break

- next jumps to the next iteration in for or while loops
- break terminates for or while loops.

```
x <- c(1, 1, 1, NA, 2)
sum <- 0
for(val in x) {
  if(is.na(val)) break
  sum <- sum + val
}
sum     ## [1] 3
x <- c(1, 1, 1, NA, 2)
sum <- 0
for(val in x) {
  if(is.na(val)) next
  sum <- sum + val
}
sum     ## [1] 5
```

Style Example: Bad

```
fWLM<-function(y,X_mat,w){T0<-t(X_mat)%*%diag(w)%*%X_mat
t<-system.time({t_1<-solve(T0)%*%t(X_mat)%*%(w*y);t2<-X_mat%*%t_
return(list(beta=t_1,hat=t2,stddev=sqrt(sum(w*(t2-y)^2))/
(length(y)-ncol(X_mat)), wts=w,t=t[[3]]))}
```

Style Example: Good

```
fit_weighted_lm <- function(response, design, weights) {
  n_obs <- length(response)
  n_coef <- ncol(design)
  time_start <- Sys.time()
  wcrossprod_design <- crossprod(design * weights, design)
  weighted_response <- weights * response
  coef <- solve(wcrossprod_design, t(design) %*% weighted_response)
  time <- Sys.time() - time_start
  fitted <- design %*% coef
  residuals <- response - fitted
  weighted_rss <- sum(weights * residuals^2)
  sd_resid <- sqrt(weighted_rss / (n_obs - n_coef))
  list(coef      = coef,
      fitted    = fitted,
      sd_resid = sd_resid,
      weights   = weights,
      time      =time)
}
```

Notation & Names

- find meaningful file names; if files need to be run in sequence, prefix them with numbers.

  `0-download.R`

  `1-parse.R`

  `2-explore.R`

- avoid uppercase
- use an underscore to separate words within a name.
- generally, variable names should be nouns and function names should be verbs.

Notation & Names

- strive for names that are concise and meaningful (this is not easy!).

```
# Good      # Bad
day_one     first_day_of_the_month
day_1       dayone
            djm1
```

- avoid using names of existing functions and variables.

```
# Bad
T <- FALSE
c <- 10
t <- temporal_variable
mean <- function(x) sum(x)
```

Formatting

- strive to limit your code to 80 characters per line.
- when indenting your code, use two spaces. Never use tabs or mix tabs and spaces.
- use <-, not =, for assignment
- place spaces around all infix operators (=, +, -, <-, etc.), before parentheses, and after comma (just like in regular English)

```
# Good
average <- mean(feet / 12 + inches, na.rm = TRUE)
# Bad
average<-mean(feet/12+inches,na.rm=TRUE)
# Good
if (debug) do(x)
plot(x, y)
# Bad
if(debug)do(x)
plot (x, y)
```

Formatting

- an opening curly brace should always be followed by a new line, while a closing curly brace should always go on its own line, unless it?s followed by else.

```
 if (y == 0) {
  log(x)
} else { y^x
}
```

- use commented lines of - and = to break up your file into easily readable chunks.

```
# Load data ---------------------------
# Plot data ---------------------------
```

- `formatR::tidy.source()` cleans up and does some automatic formatting such as consistent indent