ODE modelling in Life Sciences

INSA - 3BS 2020/2021

Introduction to the R software

Introduction

All practical session regarding introduction to the use of Ordinary Differential Equation (ODE) to model life science phenomena will be performed under the R software.

Below is the essential to know to begin with R. To learn more, see for example https://is.gd/5o755B.

What is R?

R (https://www.r-project.org/) is a free and powerful programming language for statistical computing and data visualization. R can be used to compute a large variety of draw many types of graphs including: box plot, histogram, density curve, scatter plot, line plot, bar plot,...

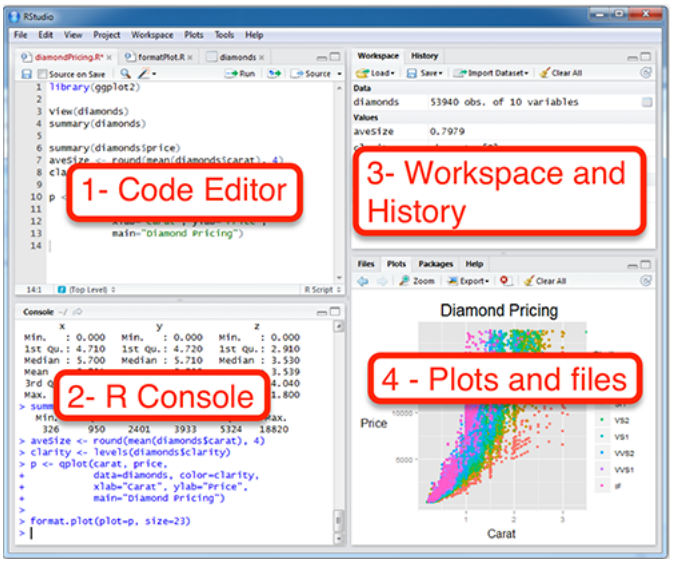
Install R and RStudio

To make things simple and user-friendly, install first R (https://www.r-project.org/) and then RStudio (https://rstudio.com/).

Here is a very useful web site to learn more about how to perform R calculations in RStudio https://rstudio.com/resources/cheatsheets/.

Running RStudio and setting up your working directory

- Launch RStudio under Windows, MAC OSX or Linux.
- Set up your working directory: change your working directory or set up a default working directory



- Close your R/RStudio session
- Functions: `setwd()`, `getwd()`

R programming basics

- Basic arithmetic operations: `+` (addition), `-` (subtraction), `*` (multiplication), `/` (division), `^` (exponentiation)

```
7 + 4 # => 11
7 - 4 # => 3
7 / 2 # => 3.5
7 * 2 # => 14
```

- Basic arithmetic functions:
  - Logarithms and exponentials: `log2(x)`, `log10(x)`, `exp(x)`
  - Trigonometric functions: `cos(x)`, `sin(x)`, `tan(x)`, `acos(x)`, `asin(x)`, `atan(x)`
  - Other mathematical functions: `abs(x)` absolute value, `sqrt(x)` square root.

```
log2(4) # => 2
abs(-4) # => 4
sqrt(4) # => 2
```

- Assigning values to variables:

```
lemon_price <- 2
```

- Basic data types: `numeric`, `character` and `logical`

```
my_age <- 28 # Numeric variable
my_name <- "Nicolas" # Character variable
#  Are you a data scientist?: (yes/no) <=> (TRUE/FALSE)
is_datascientist <- TRUE # logical variable
```

- Vectors: a combination of multiple values (numeric, character or logical)
  - Create a vector: `c()` for concatenate
  - Case of missing values: `NA` (not available) and `NaN` (not a number)
  - Get a subset of a vector: `my_vector[i]` to get the ith element

  - Calculations with vectors: `max(x)`, `min(x)`, `range(x)`, `length(x)`, `sum(x)`, `mean(x)`, `prod(x)` product of elements in `x`, `sd(x)` standard deviation, `var(x)` variance, `sort(x)`.

```
# Create a numeric vector
friend_ages <- c(27, 25, 29, 26)
mean(friend_ages) # => 26.75
max(friend_ages) # => 29
```

- Matrices: like an Excel sheet containing multiple rows and columns. Combination of multiple vectors with the same types (numeric, character or logical).
  - Create and naming matrix: `matrix()`, `cbind()`, `rbind()`, `rownames()`, `colnames()`
  - Check and convert: `is.matrix()`, `as.matrix()`
  - Transpose a matrix: `t()`
  - Dimensions of a matrix: `ncol()`, `nrow()`, `dim()`
  - Get a subset of a matrix: `my_data[row, col]`
  - Calculations with numeric matrices: `rowSums()`, `colSums()`, `rowMeans()`, `colMeans()`, `apply()`

```
matrix(data = c(5, 2, 7, 6, 4, 3, 7, 5, 4, 8, 9, 8, 9, 8, 7),
       ncol = 3, byrow = TRUE)
```

- Factors: grouping variables in your data
  - Create a factor: factor(), levels()
  - Check and convert: is.factor(x), as.factor(x)
  - Calculations with factors:
    - Number of elements in each category: summary(), table()
    - Compute some statistics by groups (for example, mean by groups): tapply()

```
# Create a factor
friend_groups <- factor(c("grp1", "grp2", "grp1", "grp2"))
levels(friend_groups) # => "grp1", "grp2"
```

```
# Compute the mean age by groups
friend_ages <- c(27, 25, 29, 26)
tapply(friend_ages, friend_groups, mean)
```

- Data frames: like a matrix but can have columns with different types
  - Create a data frame: `data.frame()`
  - Check and convert: `is.data.frame()`, `as.data.frame()`
  - Transpose a data frame: `t()`
  - Subset a data frame: `my_data[row, col]`, `subset()`, `attach()` and `detach()`
  - Extend a data frame: `$`, `cbind()`, `rbind()`
  - Calculations with numeric data frames: `rowSums()`, `colSums()`, `rowMeans()`, `colMeans()`, `apply()`

```
data.frame(name = c("Nicolas", "Thierry", "Bernard", "Jerome"),
           age = c(27, 25, 29, 26),
           height = c(180, 170, 185, 169),
           married = c(TRUE, FALSE, TRUE, TRUE))
```

- Lists: collection of objects which can be vectors, matrices or data frames
  - Create a list: `list()`
  - Subset a list
  - Extend a list

```
my_family <- list(
  mother = "Veronique",
  father = "Michel",
  sisters = c("Alicia", "Monica"),
```

```
  sister_age = c(12, 22)
  )
# Print
my_family
```

Getting help with functions in R programming

- Getting help on a specific function: `help(mean)`, `example(mean)`
- General help about R: `help_start()`
- Others functions: `apropos()` and `help.search()`

Installing and using R packages

An R packages is an extension of R containing data sets and specific functions to answer specific questions.

- Installing R packages
  - Install a package from CRAN: `install.packages()`
  - Install a package from Bioconductor: `biocLite()`
  - Install a package from GitHub: `devtools::install_github()`
  - View the list of installed packages: `installed.packages()`
  - Folder containing installed packages: `.libPaths()`
- Load and use an R package: `library()`
- View loaded R packages: `search()`
- Unload an R package: `detach(pkg_name, unload = TRUE)`
- Remove installed packages: `remove.packages()`
- Update installed packages: `update.packages()`

R base functions for importing data

The R base function `read.table(file, sep, header, dec)` is a general function that can be used to read a file in table format. The data will be imported as a data frame.

- `file`: the path to the file containing the data to be imported into R.
- `sep`: the field separator character. `\t` is used for tab-delimited file.
- `header`: logical value. If `TRUE`, `read.table()` assumes that your file has a header row, so row 1 is the name of each column. If that is not the case, you can add the argument `header = FALSE`.
- `dec`: the character used in the file for decimal points.

```
# Read tabular data into R
read.table(file, header = FALSE, sep = "", dec = ".")
# Read "comma separated value" files (".csv")
read.csv(file, header = TRUE, sep = ",", dec = ".", ...)
# Or use read.csv2: variant used in countries that
# use a comma as decimal point and a semicolon as field separator.
read.csv2(file, header = TRUE, sep = ";", dec = ",", ...)
# Read TAB delimited files
read.delim(file, header = TRUE, sep = "\t", dec = ".", ...)
read.delim2(file, header = TRUE, sep = "\t", dec = ",", ...)
```

It is possible to choose a file interactively using function `file.choose()`, which is recommended if you are a beginner in R programming:

```
# Read a txt file
my_data <- read.delim(file.choose())
# Read a csv file
my_data <- read.csv(file.choose())
```

At last, it is possible to use functions `read.delim()`, `read.csv()` and `read.table()` to import files from the web.

```
my_data <- read.delim("http://www.sthda.com/upload/boxplot_format.txt")
# Print the first 6 rows
head(my_data)
```

R built-in data sets

- List of pre-loaded data
- Loading a built-in R data
- Most used R built-in data sets
  - `mtcars`: Motor Trend Car Road Tests
  - `iris`
  - `ToothGrowth`
  - `PlantGrowth`
  - `USArrests`

Graphics with R

- Import data into R, either your own data or built-in data

```
# Create my_data
my_data <- mtcars
head(my_data, 6)
```

- The R base function `plot()` can be used to create graphs.

```
plot(x = my_data$wt, y = my_data$mpg,
     pch = 16, frame = FALSE,
     xlab = "wt", ylab = "mpg", col = "#2E9FDF")
```

- Save your graph: `Plots` bottom right panel in RStudio, then `Export`, then `Save as Image` or `Save as PDF`. It is also possible to save the graph using R codes as follow:
  - Specify files to save your image using a function such as `jpeg()`, `png()`, `svg()` or `pdf()`. Additional argument indicating the width and the
  - Create the plot
  - Close the file with `dev.off()`

```
# 1. Open jpeg file
jpeg("rplot.jpg", width = 350, height = "350")
# 2. Create the plot
plot(x = my_data$wt, y = my_data$mpg,
     pch = 16, frame = FALSE,
     xlab = "wt", ylab = "mpg", col = "#2E9FDF")
# 3. Close the file
dev.off()
```

Plotting functions

Plotting functions are divided into two basic groups:

1. **High-level** plotting functions create a new plot on the graphic right panel, possibly with axes, labels, titles,...

Examples: `plot()`, `curve()`, `barplot()`, `boxplot()`...

```
curve(x^2, from = -2, to = 2, las = 1)
```

2. **Low-level** plotting functions add more information to an existing plot, such as extra points, lines and labels.

Examples: `points()`, `lines()`, `legend()`...

```
curve(x^2, from = -2, to = 2, las = 1)
points(x = c(-2, -1, 0, 1, 2),
       y = c(4, 1, 0, 1, 4), pch = 19)
```

Remark: function `curve()`is both a high- and a low-level function, given the use of option `add = TRUE`.

```
curve(x^2, from = -2, to = 2, las = 1)
curve(2 * x^2, add = TRUE, col = "red")
```

Graphs in R 2.0

Today, there is a wide use of pakcage `ggplot2` to create graphs. Package `ggplot2` is based on the grammar of graphics, tha is the idea that you can build every graph from the same components: a data set, a coordinate system and geoms—visual marks that represent data points. To learn more: https://is.gd/2bGhyk.