Force data analysis with R studio and Scilab software

Table of Contents

## 1. Experimental context and data

Two groups of individuals were exposed to a model of muscle conditioning, dry immersion, for 5 days (DI5j). One group was a control group (CTR), while the other group wore compression sleeves (TC).

Isometric maximum force measurements (MVC) were carried out in precondition (BDC-4) four days before DI5j, and in postcondition (R + 1) one day after DI5j.
Measurements were made for contractions of about 5 seconds, in extension and flexion, for the ankle and knee, with a Cybex machine.

The files of raw data are from the Cybex machine in .CTM (data), .CTA (information about .CTM files),.mbg (setup) and from operators in .txt (repartitions of the subjects in groups) format.

## 2. Objectives

The point of this study is to see the impact of the DI5j on the groups and to compare them. The variables to extract and observe from this data are the parameters k1 and kACT of the ascent of force during the MVC.

Those parameters emerge from a mathematical model formula based on a sigmoid form, used to understand physical phenomena in the human body. Actually, the study which proves the effectiveness of its use for this type of data is being written by researchers from DMEM team, INRAE, France.

Hypothesis concerning those parameters are that they will decrease for the CRT group in post DI5j in comparison to pre DI5J, and no changes for TC group.

## 3. Directory structure & order of file execution

To facilitate use and minimize potential errors, the directory structure is:

- 'WRK': working directory for the data analysis problem
  - 'DAT': data used as input for analyses
  - 'PRG': R and Scilab scripts and functions for the analyses
    - 'R_functions': R functions for the analyses
    - Scilab_functions': Scilab functions for the analyses
  - 'RES': results of R and Scilab analyses
    - 'RES_R': results of R analyses
      - 'Plots_R': plots of results of R analyses
    - 'RES_Scilab': results of Scilab analyses
      - 'Plots_Scilab': plots of results of Scilab analyses

Functions are almost always in an independent file which contains only the function and its description.

Files should be uses in the following order:

- main_P1_tri_correction_paquets.R
- main_P2_analyses_force.sce
- main_P3_STATS_force.R

# 4. Part 1: Correction of identification data, creation of sub-packages in R studio (main_P1_tri_correction_paquets.R)

This file consists of two parts: an initialization part of the working environment, and a working part.

## 4.1. Initialize

The rm(list = ls()) function is used to clear the workspace.
Then the directory structure is code, relative to the present file.
The dirname(rstudioapi::getSourceEditorContext()$path) function gets the absolute file path of P_STATS_force.R file. The setwd(dirname(PRG_PATH)) function gets the absolute file path of PRG_PATH as the new working directory. WRK_PATH is defined as the present directory with the getwd() function. Others paths are defined with the file.path function.
To load different packages used in the analysis, the install.package() and the library() functions, following the name of the package are used.

## 4.2. STEP 1: list the raw files in .CTM format

The list.files() function is used to put in a variable the full names of every files with the pattern ".CTM".
The names of the variables of interest are listed in Variables. Length of variables Variables and CTMFileList are obtain with the length function and use in the data.frame() function as ncol and nrow arguments to create the dataframe Tableau_recap, which will be used to compile informations. Columns of Tableau_recap are rename with variables name of Variables using colnames() function.

## 4.3. STEP 2: extract important informations

To work on each file listed in Tableau_recap one after the other, an if-end loop is open, with a vector i.
Length of vector i is equal to the Nb_file (equal to number of rows in Tableau_recap).
The read.table() function load file in Data as character categories, using file path in CTMFileList at i value (as a character using as.character() function).
Important data are put in variables using coordinates (known by the programmer) and then all variables are compiled in Tableau_recap.
Each columns are converted in the right format, using as.factor(), as.Date() and as.numeric() functions.

## 4.4. STEP 3: Identification and correction of potential errors in files

Using the summary() function, principal statistiques of Tableau_recap are made and shown in console with the print() function.
The programmer made observations which are shown in console before being corrected in Tableau_recap_corrige, using is.na() function and '%>%' manipulation (from "dplyr" package).
By the same way as before, the summary of Tableau_recap_corrige is made and shown in console to note changes.

## 4.5. STEP 4: save Tableau_recap_corrige in .csv file in RES

FileName_Tableau is defined, and used to define FullFileName_Tableau with the file.path() function and the RES_R_PATH. Then the write.table() function allows to save Tableau_recap_corrige in RES_R in a .csv file.

A message, made with the paste0() function to concatenate vectors and text after converting to character, which recapitulate the file saves in RES_R is shown in the console.

### 4.6. STEP 5: create sub-packages of data depending on conditions

Sub-packages are done only for the condition MVC (Program = 6s), not for the condition Fatigue (Program = 90s)
In a first time, conditions are listed for each category (Body Part and Movement). Then two loops are open to work on the different combinations of conditions (each loop work with a vector of the length of each category), and every row of Tableau_recap_corrige corresponding to those combination are put in Paquet_MVC_Body_Part before being saved.
FileName_Paquet is defined, and used to define FullFileName_Paquet with the file.path() function and the RES_R_PATH. Then the write.table() function allows to save Tableau_recap_corrige in RES_R in a .csv file.
A message, made with the function paste0() to concatenate vectors and text after converting to character, which recapitulate the file saves in RES_R is shown in the console.

## 5. Part 2: Analyses of force data with Scilab (main_P2_analyses_force.sce)

Strength data analysis with Scilab are done in the main.sce file. This file consists of two parts: an initialization part of the working environment, and a working part.

### 5.1. Initialize

For initializing the environment, the get_absolute_file_path function is used to retrieve the path to the main.sce file as the path to the programs. After which thanks to the fullfile function, the path of the InitTRT.sce file is defined, before being executed with the exec function.
As some documents contain several sub-documents, these are added, using the paths designed in the InitTRT.sce file, and thanks to the fullfile function.

#### 5.1.1. InitTRT.sce

It is the file InitTRT.sce which will allow to clean the working environment and to obtain all the paths of the various documents contained in the document WRK.
This file is the Version 2.1.0, made in Oct 10, 2019 by D. Mottet, and has been find at: https://github.com/DenisMot/ScilabDataAnalysisTemplate

### 5.2. STEP 1: Define sampling frequency and units

Sampling frenquency (Fe) is gave by operator to be used for creating time vector later.
Units of variables are defined to be used for plots later.

### 5.3. STEP 2: Define conditions to work on (Body_Part, Movement) & open right file

Beforehand, variables Body_Part and Movement are created as empty vectors.
Two while loop are open to ask to the user, with the input function, the conditions he wants to work on (Body_Part, Movement).
while-end function is used to open a loop to execute instructions as long as the wanted conditions is respected. For example, here a condition is : if the vector Body_Part is different of “Cheville” or “Genou”, then ask for input.
input function gives the user the prompt in the text string and then waits for input from the keyboard.

FileName is the variable containing the name of the file, composed with the chosen conditions. The complete file name is constructed with fullfile() function with the RES_R_PATH.
Then the corresponding file is load in the Tableau_recap variable thanks to csvRead() function.
Headers and data are separated, as Headers are in the first line of Tableau_recap, and data are the rest.

Finally, disp() function is used to displays messages and variables to resume the chosen conditions and the open file.

### 5.4. STEP 3: Prepare data for limits detection

To work on each file listed in Tableau_recap one after the other, an if-end loop is open, with a vector i.
Length of vector i is equal to the number of rows in the variable Tableau_recap, obtain with size() function ('r' is to count the number of lines).

fscanfMat() function load file in Data variable, using file path in the first column of Tableau_recap, at row i.

Force data are extract from Data, as known being in its first column.
Vector Time is create using number of rows in Force (size) and sampling frequency (Fe). (NB: Time vector must be reduced of 1 value to have the same size as Force vector)

In case the signal is not in the right way, condition checked with the if loop which verify if the sum (using sum() function) of Force is greater than zero, the data are multiplied by -1 on each point.

To help the detection of limits the sampling of Force is reduced with the intdec() function, with a ratio defined by the programmer. The new variable is called Force_M.
As made for the Force vector, the Time_M vector is created, using the same ratio as for Force_M (sampling frenquency is divided by the ratio).

The CLEANING operator function is used to remove artifacts in the data by replacing them by values zero. The new variable is called Force_M_C.

Index and value of the max of Force_M_C are find with the max() function.
Then, the part of the curve from the first value of Force_M_C to its firct max (first value of i_Max_Force_M_C(1)) is extract in Force_M_C_p1 to facilitate the next operations.

Derivatives of Force_M_C,Time_Diff_Force_M and Force_M_C_p1 are calculated with the diff() function.

Index and value of the max of Diff_Force_M_C_p1are find with the max() function.

#### 5.4.1. CLEANING.sci

CLEANING function is used to obtain as output the same vector of data as in input but cleaned, meaning without the artefacts before and after the principal part of the curve.

Once the function is open, index and value of the max of Signal are find with the max() function. Then Signal is cut in two parts (P1, P2) at its max (in case of multiples max it is the first index which is used).
The threshold value (TS) is defined at 1. With the find() function, every index of value under the threshold value are reported in vector Starts for P1 and in Ends for P2.
Using an if-end loops in case of no values in Starts, every value before the last value under TS are replace by zero; same goes for P2: every value after the first value under TS are replace by zero.
Finally, P1 and P2 are concatenated to give the final vector Signal_clean.
To be sure that Signal_clean and Signal have the same dimensions, an if-end loop compares the numbers of row to the numbers of columns in Signal (using size() function with 'r' and 'c') to apply the right concatenation.
As P1 and P2 share the max value, P2 need to be taken from its 2nd value.

### 5.5. STEP 4: Detection of ASCENT OF FORCE phase limits

The start of Ascent corresponds to the last value of Diff_Force_M_C_p1 equal or under zero.
In a first time we try to find with the find() function the index of values under or equal to 0 in Diff_Force_M_C_p1 before its max.
With an if loop we check if there is a value in i_zero, if yes i_Start_ascent is equal to the last value in i_zero.
If no, i_Start_ascent is equal to 1.
Then Time_Start_ascent is the time value at the i_Start_ascent index in Time_M.

The end of Ascent corresponds to the first value of Diff_Force_M_C_p1 equal or under zero after i_Start_ascent, because when the derivative pass through zero, there is a change in the direction of the signal.
In a first time we try to find with the find() function the index of values under or equal to 0 in Diff_Force_M_C_p1 after i_Start_ascent.
With an if loop we check if there is a value in i_zero_diff, if yes i_End_ascent is equal to i_Start_ascent plus the first value in i_zero_Ascent.
If no, i_End_ascent is equal to the index of last value of Ascent.
Then Time_End_ascent is the time value at the i_ End _ascent index in Time_M.

### 5.6. STEP 5: Plotting Signal with limits

The figure() function open a graphic window where plots will appear. Here the number of the figure is the value in i during the loop.
The subplot() function virtually grids the figure and sets the plotting area to a chosen cell. Here we chose to plot in the upper part, with the plot() function, Force as a function of Time in red ('r' argument) and Diff_Force_M_C as a function of Time_Diff_Force_M in blue ('b' argument).
With an if loop, we ckeck if values ware found for Time_Start_ascent or Time_End_ascent. If no, a message saying the number of the curve which could not been done is shown in console with disp() function, and Time_Start_ascent or Time_End_ascent take value zero. If yes, limits are added to the plot as yellow vertical lines.

The legend() function allows to add a legend of the different plots, and the xtitle() function allows to add a title to the plot, which has been defined according to the chosen conditions and the corresponding name subject, and axis name.

### 5.7. STEP 6: Modelization of ascent of force

First, with an if loop we ckecks if Time_Start_ascent or Time_End_ascent are equal to zero. If yes, kACT, k1 and R2 take value zero.
If no, Start and End are defined as the closest value of Time_Start_ascent or Time_End_ascent respectively in Time, using find() function.
Then Ascent is defined as the part of Force data from the first value in Start to the first value in End.
As for Time, Time_Ascent is created using N_Ascent, get with size() function, and sampling frequency (Fe).

The ASCENT_CURVE_SIGM operator function is used to find the value of k1 and kACT parameters, and the R2 value (coefficient of correlation between Ascent and model), with as input Ascent and the sampling frequency (Fe).

Thanks to this and to the SIGM operator function, the modelization of Ascent, called Theorical_Ascent is calculated.

#### 5.7.1. ASCENT_CURVE_SIGM.sci

ASCENT_CURVE_SIGM function is used to obtain as output the two parameters (k1, kACT) of the signal modelization (using SIGM function) and the coefficient of correlation R2 between the signal and the modelization, using as input the signal ("Ascent") and the sampling frequency (Fe).
Vector Time is create using number of rows in Ascent (size) and sampling frequency (Fe). (NB: Time vector must be reduced of 1 value to have the same size as Ascent vector)

Then initial parameters for calculation are defined:

- N_1 as a vector full one 1, with the same size as Ascent;
- k1_0 as the first value of k1 for calculation (equal to the maximum of Ascent less 10);
- kACT_0 as the first value of kACT for calculation (arbitrarily set to 0.001);
- factors_0 as the vector containing k1_0 and kACT_0.

The myfun function is exceptionally not being put in another file for better understanding and because it is always used in this context. It calculates the vector containing the differences between theoretical and real data at each point. It is defined as in ‘help leastsq’ and uses SIGM operator function.
The leastsq function is used to solves non-linear least squares problems. It is defined as in ‘help leastsq’ and uses myfun function.

Finally, k1 and kACT values are exctract from xopt, and the coefficient of correlation (R2) between Ascent and Theorical_Ascent is calculated with the squared value of the correl function.

#### 5.7.2. SIGM.sci

SIGM function is used to obtain as output a vector “y” of data calculated on a sigmoïde base, using as input a time vector and a parameters vector (k1, kACT).

Once the function is open, the length of vector time is calculated to define the vector for a for-end loop, which calculate y value on each point of time. The use of the abs function in formula is to avoid complex number problems.

### 5.8. STEP 7: Plotting Signal with model

Here we chose to plot in the lower part, with the subplot() function in the last figure open, with the plot() function, Ascent as a function of Time_Ascent in red ('r' argument) and Theorical_Ascent as a function of Time_Ascent in green ('g' argument)
The legend() function allows to add a legend of the different plots, and the xtitle() function allows to add a title to the plot, which has been defined according to the chosen conditions and the corresponding name subject, and axis name.

### 5.9. STEP 8: Save plot in pdf

FileNamePlot is composed with the chosen conditions and the value of vector i.
FullFileNamePlot is constructed with fullfile() function with the RES_SCILAB_PATH.
Then the plot is saved thanks to xs2pdf() function.

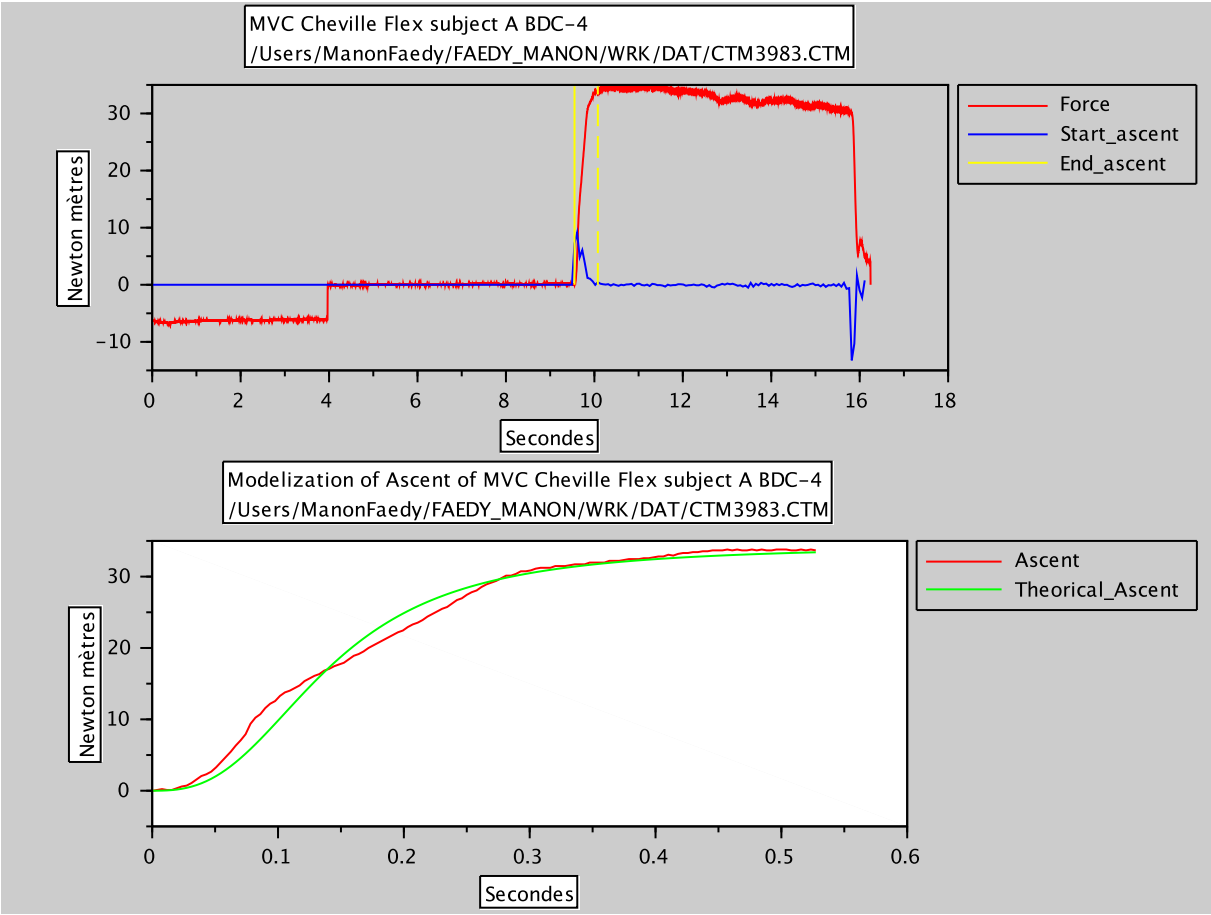


Fig. 1: Example of plot save as PDF

### 5.10. STEP 9: Compile results

Time_Start_ascent, Time_End_ascent, k1, kACT and R2 are added as string in Tableau_recap_resultats.
Every vectors/matrix used during the loop are clear to remove data from the previous loop.
Headers are completed and add Tableau_recap_resultats.

### 5.11. STEP 10: Save results as a csv file in RES_SCILAB repertory

FileNameTableau_recap_resultats is composed with the chosen conditions. FullFileNameTableau_recap_resultats is constructed with fullfile() function with the RES_SCILAB_PATH.
Then Tableau_recap_resultats is saved thanks to csvWrite () function.

## 6. Part 3: Sorting, statistics and plots of data with R studio (main_P3_STATS_force.R)

This file consists of two parts: an initialization part of the working environment, and a working part.

### 6.1. Initialize

The rm(list = ls()) function is used to clear the workspace.
Then the directory structure is code, relative to the present file.
The dirname(rstudioapi::getSourceEditorContext()$path) function gets the absolute file path of P_STATS_force.R file. The setwd(dirname(PRG_PATH)) function gets the absolute file path of PRG_PATH as the new working directory. WRK_PATH is defined as the present directory with the getwd() function. Others paths are defined with the file.path function.
To load different packages used in the analysis, the install.package() and the library() functions, following the name of the package are used.

### 6.2. STEP 1: Define conditions to work on (Body_Part, Movement) & load data

Beforehand, variables Body_Part and Movement are created as empty variables.
Two while loop are open to ask to the user, with the readline function, the conditions he wants to work on (Body_Part, Movement).
while-end function is used to open a loop to execute instructions as long as the wanted conditions is respected. For example, here a condition is: if the vector Body_Part is different of “Cheville” or “Genou”, then ask for input.
readline function gives the user the prompt in the text string and then waits for input from the keyboard.

FileNameData is the variable containing the name of the file, composed with the chosen conditions. The complete file name is construct with file.path function with the RES_SCILAB_PATH.
Then the corresponding file is load in the Data dataframe thanks to read.delim function.

Finally, print function is used to displays messages and variables to resume the chosen conditions and the open file, concatenate with paste0 function.

### 6.3. STEP 2: Group data and delete unused ones

In a first time, data are filtered: Data_filt get values from Data except rows for which the R2 value is below 0.95.

Then, the aggregate() function (from “stats” package) splits the data into subsets, computes summary statistics for each, and returns the result the dataframe SUM. As some columns were not numeric, the message ‘There were 50 or more warnings (use warnings() to see the first 50)’ will be return in the console. After changing columns names with colnames() function, data from columns "Sujet", "Moment", "k1", "kACT") are put in the Data_filt_sum data frame.

>%> group_by() function is used to make a summary of Data_filt_sum to see if some subjects have data only in pre or post. With a for loop, to work on every row of SUM_subjects, an if loop checks if a subject has more than one row, and if no it delete this subject in the copy of Data_filt_sum (Data_filt_sum_filt), and a message saying which subjects have been deleted appears in the console (using paste0() and print() functions).

### 6.4. STEP 3: Add group factor and separate variable

The file containing distributions of subjects in groups is load in the Rep_grp dataframe thanks to read.delim2 function. (A previously, the complete file name is construct with file.path function with the RES_SCILAB_PATH.)

To deleted rows with subjects which are not in the data, a for loop and an if loop allows to checks for every rows in Rep_grp if the subject is present in Data_filt_sum_filt, if no rows are filled with NaN and then deleted with the complete.cases function (from “stats” package). Also, a message saying which subjects have been deleted appears in the console (using paste0() and print() functions).

Data in the columns “Groupe” of Rep_grp are extract as character in the variable Groupe, using as.character() function. A column is added to the dataframe Data_filt_sum_filt next to the column “Sujet” with the add_column() function (from “tibble” package), and filled with twice the data in Groupe (converted in factor with the as.factor() function).

A dataframe is made for each variable, one for Data_k1 and one for Data_kACT, and their column 4 are rename "Variable" using colnames function, to be used after in "STATS_ANOVA.R” function.

### 6.5. STEP 4: Statistics and save plots & results

The statistical test for data with two measurement times (BDC-4 vs R+1) and two groups (CTR vs TC) to compare is a Repeated Measures ANOVA. The STATS_ANOVA operator function is made for this type of analysis.

Using file.path() function the FullFileName for the operator function "STATS_ANOVA.R" is made, with the PRG_R_FUNCTION_PATH, before being loaded with the source() function.

The FileNamePlotPDF for the PDF file containing plots is made with paste0() function to have a name adapted to the chosen conditions, and the FullFileNamePlotPDF is made with the file.path() function and the RES_R_PLOTS_PATH.
PDF file is open with the pdf function (from “grDevices” package) and will save every plots until dev.off() (from “grDevices” package) function close it.

The STATS_ANOVA function is executed with Data_k1 and after with Data_kACT.

Then, messages and results from the "STATS_ANOVA.R" function are shown in the console using print() function, before being saved in a txt file using capture.output() function (from “utils” package). As for FileNamePlotPDF, FileNameResis made with

paste0() function to have a name adapted to the chosen conditions, and the FullFileNameResis is made with the file.path() function and the RES_R_PATH.

#### 6.5.1. STATS_ANOVA.R

The calling sentence for this function is “STATS_ANOVA = function(dataframe, Variable_name)”.
Dataframe is a dataframe containing four columns: “Subject” (factor 2 levels), “Groupe” (factor 2 levels), “Moment” (factor 2 levels), “Variable” (numeric).
Variable_name is the name of the variable of interest as a character string.
STATS_ANOVA is a list of the analysis results.
This function provides: descriptive statistics, pre Anova tests, Anova test, Post Hoc tests, size effect and boxplots.

##### 6.5.1.1. Initialize

To load different packages used in the analysis, the install.package() and the library() functions, following the name of the package are used.

##### 6.5.1.2. STEP 1: Descriptive stats

%>% group_by() (from ‘dplyr’ package) and the get_summary_stats() (from ‘rstatix’ package) functions are used to make summary of type "mean_sd" by Moment and by Groupe factors.

##### 6.5.1.3. STEP 2: Pre-anova tests

Equality of variance (Levene test) of Variable between Groupe by Moment is calculated with %>% group_by() (from ‘dplyr’ package) and levene_test() (from ‘rstatix’ package) functions.

Normality assumption (Shapiro-Wilk test) of Variable is calculated with %>% group_by() (from ‘dplyr’ package) by Moment and by Groupe factors.

##### 6.5.1.4. STEP 3: Anova

An anova is made using anova_test() function (from ‘rstatix’ package), with the formula Variable ~ Moment * Groupe, Sujet as wid argument, 3 as type argument because data are unbalanced, TRUE as white.adjust argument to correct heteroscedasticity and TRUE as detailed argument to have more informations in the anova table.

##### 6.5.1.5. STEP 4: Post-Hoc

Pairwise comparisons between groups or times are made with %>% group_by() (from ‘dplyr’ package) and %>% pairwise_t_test() (from ‘rstatix’ package) functions, using FALSE as paired argument for comparisons between groups, and TRUE for comparisons between times, and using "bonferroni" as p.adjust.method argument. One is made for Moment factor and one for Groupe factor.

##### 6.5.1.6. STEP 5: Size effect (Cohen's d)

The same process as for Post-Hoc tests is made for size effect calculations (Cohen's d), using %>% group_by() (from ‘dplyr’ package) and cohens_d() (from ‘rstatix’ package) functions. One is made for Moment factor and one for Groupe factor.

##### 6.5.1.7. STEP 6: Box plots with p-values

The get_y_position function (from ‘rstatix’ package is used on each pairwise test tables to autocompute p-valuepositions for plotting significance, the x argument correspond to the comparison factor.

Then, boxplots are made with the ggboxplot() function (from ‘ggpubr’ package). Some important arguments to put in light are : the title argument which is made using paste0() function and Variable_name to adapt the title, and same for ylab argument, and add argument which is “jitter” to represent subject as points in addition to the boxplot.

P values are added on the boxplot with the stat_pvalue_manual() function (from ‘ggpubr’ package), in which one the pairwise tests are put. It uses "p = {p}" as label argument to add “p =” before the p value. The y.position argument uses the “y.position”, added before to the pairwise test tables.

To add more information on the plots, labs() function (from ‘ggplot2’ package) with get_test_label() function (from ‘rstatix’ package) as subtitle argument and get_pwc_label() function (from ‘rstatix’ package) as caption argument.

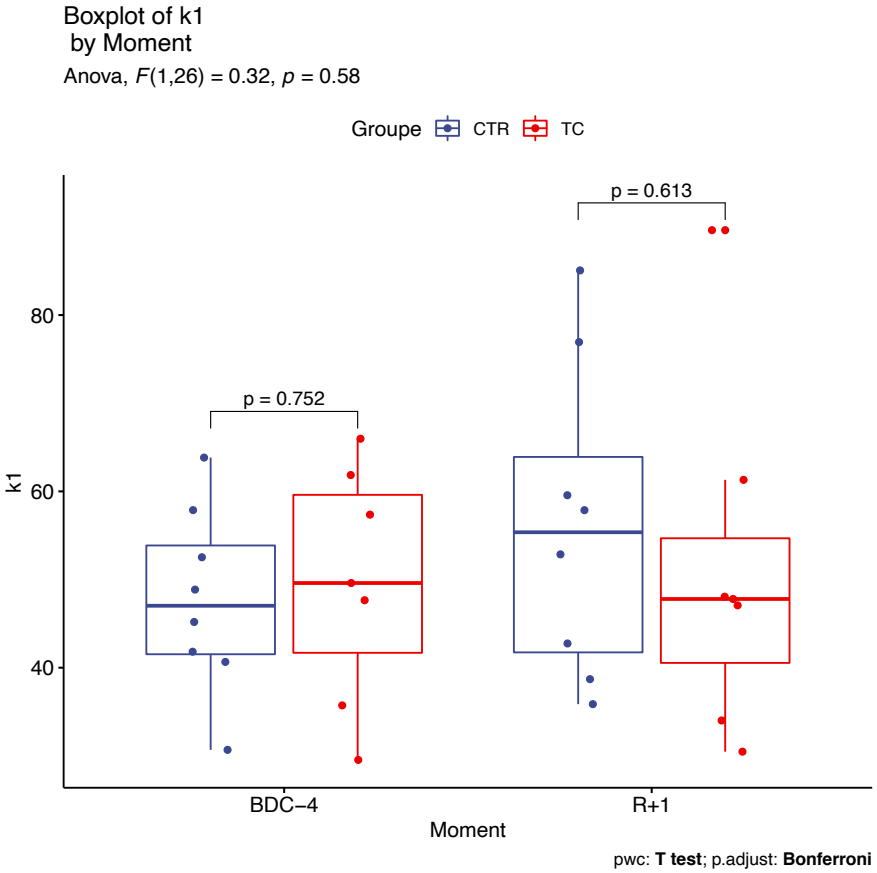


Fig.2: Example of boxplot save as PDF (Conditions : Cheville – Flex)

#### 6.5.1.8. STEP 7: Output

To organize results gave in output of the STATS_ANOVA operator function, the list function is used name and store variables.