Tutorial – Comparing means under controlled conditions

Diplôme Universitaire en Bioinformatique Intégrative (DUBii)

Jacques van Helden

2020-03-12

Contents

Required libraries

Load the following libraries or install them if required.

```
require(knitr)
```

Introduction

This tutorial aims at providing an empirical introduction to the application of mean comparison tests to omics data.

The goals include

- revisiting the **basic underlying concepts** (sampling, estimation, hypothesis testing, risks. . .);
- perceiving the problems that arise when a test of hypothesis is applied on several thousand of features (**multiple testing**);
- introducing some methods to circumvent these problems (**multiple testing corrections**);
- using graphical representations in order to grasp the results of several thousand tests in a winkle of an eye:
    - p-value histogram
    - MA plot

  - volcano plot

The whole tutorial will rely on artificial data generated by drawing random numbers that follow a given distribution of probabilities (in this case, the normal distribution, but other choices could be made afterwards).

The tutorial will proceed progressively:

1. Generate a multivariate table (with *individuals* in columns and *features* in rows) and fill it with random data following a given distribution of probability.
2. Measure different descriptive parameters on the sampled data.
3. Use different graphical representations to visualise the data distribution.
4. Run a test of hypothesis on a given feature.
5. Run the same test of hypothesis on all the features.
6. Use different graphical representations to summarize the results of all the tests.
7. Apply different corrections for multiple testing (Bonferroni, Benjamini-Hochberg, Storey-Tibshirani q-value).
8. Compare the performances of the test depending on the chosen multiple testing correction.

Experimental setting

Well, by "experimental" we mean here that we will perform *in silico* experiments.

Let us define the parameters of our analysis. We will generate data tables of artificial data following normal distributions, with either different means (tests under H_1) or equal means (tests under H_0).

We will do this for a number of features $m_0 = 10,000$ (number of rows in the "H_0" data table), which could be considered as replicates to study the impact of sampling fluctuations.

In a second time (not seen here) we could refine the script by running a sampling with a different mean for each feature, in order to mimic the behaviour of omics datasets (where genes have different levels of expression, proteins and metabolite different concentrations).

Parameters

Parameter	Value	Description
n_1	$2, 3, 4, 8, 16, 32, 64$	size of the sample from the first population. individual choice. Each participant will choose a given sample size
n_2	$= n_1$	size of the sample from the second population
μ_1	10 or 7	mean of the first population. each participant will chose one value
μ_2	10	mean of the second population
σ_1	2	Standard deviation of the first population
σ_2	3	Standard deviation of the second population
m_0\$	$10,000$	number of features under null hypothesis

Sample sizes

Each participant will choose a different sample size among the following values: $n \in 2, 3, 4, 5, 8, 16, 64$. Noteowrthy, many omics studies are led with a very small number of replicates (frequently 3), so that it will be relevant to evaluate thee impact of the statistical sample size (number of replicates) on the sensibility (proportion of cases declared positive under H_1).

Performances of the tests

We will measure the performances of a test by running $r = 10,000$ times under H_0, and $r = 10,000$ times under H_1.

- count the number of FP, TP, FN, TN
- compute the derived statistics: FPR, FDR and Sn

$$FPR = \frac{FP}{m_0} = \frac{FP}{FP + TN}$$

$$FDR = \frac{FP}{\text{Positive}} = \frac{FP}{FP + TP}$$

$$Sn = \frac{TP}{m_1} = \frac{TP}{TP + FN}$$

$$PPV = \frac{TP}{\text{Positive}} = \frac{TP}{TP + FP}$$

Recommendations

Coding recommendations

1. Choose a consistent coding style, consistent with a reference style guide (e.g. Google R Syle Guide). In particular:
   - For variable names, use the camel back notation with a leading lowercase (e.g. `myDataTable`) rather than the dot separator (`my.data.table`)
   - For variable names, use the camel back notation with a leading uppercase (e.g. `MyMeanCompaTest`).
2. Define your variables with explicit names (sigma, mu rather than a, b, c, …).
3. Comment your code
   - indicate what each variable represents
   - before each segment of code, explain what it will do
4. Ensure consistency between the code and the report $\rightarrow$ inject the actual values of the R variables in the markdown.

Scientific recommendations

1. Explicitly formulate the statistical hypotheses before running a test.
2. Discuss the assumptions underlying the test: are they all fulfilled? If not explain why (e.g. because we want to test the impact of this parameter, …) .

Tutorial

Part 1: generating random datasets

Define your parameters

Write a block of code to define the parameters specified aboce.

Note that each participant will have a different value for the sample sizes (n_1, n_2).

```
#### Defining the parameters ####

## Sample sizes.
## This parameter should be defined individually for each participant
n1 <- 16 # sample size for the first group
n2 <- 16 # ssample size for second group

## First data table
m <- 1000 # Number of features
mu1 <- 7 # mean of the first population
mu2 <- 10 # mean of the second population

## Standard deviations
sigma1 <- 2 # standard deviation of the first population
sigma2 <- 3 # standard deviation of the second population

## Significance threshold.
## Note: will be applied successively on the different p-values
## (nominal, corrected) to evaluate the impact
alpha <- 0.05
```

The table below lists the actual values for my parameters (your values for n_i will be different).

Parameter	Value	Description
μ_1	7	Mean of the first population
μ_2	10	Mean of the second population
σ_1	2	Standard deviation of the first population
σ_2	2	Standard deviation of the second population
n_1	16	Sample size for the first group
n_2	16	Sample size for the second group

Generate random data sets

Exercise:

- Generate an data frame named `group1` which with m_0 rows (the number of features under H_0, defined above) and n_1 columns (sample size for the first population), filled with random numbers drawn from the first population.
- Name the columns with labels indicating the group and sample number: `g1s1`, , `g1s2` … with indices from 1 n_1.
- Name the rows to denote the feature numbers: `ft1`, `ft2`, … with indices from 1 to m_0.
- Check the dimensions of `group1`.
- Print its first and last rows to check its content and the row and column names.

- Generate a second data frame named `group2` for the samples drawn from the second population with the appropriate size, and name the columns and rows consistently.

Useful functions

- `rnorm()`
- `matrix()`
- `data.frame()`
- `paste()`
- `paste0()`
- `colnames()`
- `rownames()`
- `dim()`

Trick:

- the funciton `matrix()` enables us to define the number of columns and the number of rows,
- the function `data.frames()` does not enable this, but it can take as input a matrix, from which it will inherit the dimensions

Solution (click on the "code" button to check the solution).

```
#### Generate two tables (one per population) with the random samples ####

## Info: for didactiv purpose we will use a progressive approach to generate
## the data the first group,  and a more compact formulation (in one command)
## for the second group.

## Dataset under H0, with a progressive approach

## Generate a vector of size m*n1 with all the random values
## for each feature and each infividual
normVector <- rnorm(n = m * n1, mean = mu1, sd = sigma1)

## Wrap the vector in an m x n1 matrix
normMatrix <- matrix(data = normVector,
                     nrow = m,
                     ncol = n1)

## Create a data frame with the content of this matrix
group1 <- data.frame(normMatrix)

## Set the column and row names
colnames(group1) <- paste(sep = "", "g1s", 1:n1)
rownames(group1) <- paste(sep = "", "ft", 1:m)

## Dataset under H1: use a more direct approach, compact the 3 steps in a single command
group2 <- data.frame(
  matrix(data = rnorm(n = m * n2, mean = mu2, sd = sigma2),
         nrow = m,
         ncol = n2))
colnames(group2) <- paste(sep = "", "g2s", 1:n2)
rownames(group2) <- paste(sep = "", "ft", 1:m)
```

Check the content of the data table from the first group.

```
dim(group1)
```

```
[1] 1000   16
```

```
kable(head(group1))
```

	g1s1	g1s2	g1s3	g1s4	g1s5	g1s6	g1s7	g1s8	g1s9	g1s10	
ft1	8.810784	7.904091	7.499375	7.968559	5.126577	8.327520	5.485461	7.206176	9.212694	5.386134	
ft2	7.410872	9.706472	7.030511	9.229786	3.930087	10.063132	8.165548	5.718797	5.969557	10.369197	
ft3	4.315892	6.169750	2.986875	8.462985	9.350682	1.713240	5.021550	7.004814	3.662523	7.322640	
ft4	7.816991	4.555216	7.126701	6.406219	7.635769	6.629055	2.899936	7.412709	9.298143	7.608633	
ft5	9.307010	7.366520	7.239244	7.342546	7.108199	5.596629	8.063446	5.518158	7.286379	7.858502	
ft6	9.106303	6.937542	8.788831	2.863629	12.349995	9.947563	6.614870	6.126687	5.733720	9.224333	1

```
kable(tail(group1))
```

	g1s1	g1s2	g1s3	g1s4	g1s5	g1s6	g1s7	g1s8	g1s9	g1s10
ft995	5.909896	5.709044	8.546074	6.450778	6.750382	5.386992	5.644113	5.870854	2.180627	6.200203
ft996	3.460329	9.463328	8.043028	6.563592	4.617538	9.717411	4.735573	6.355436	3.798099	8.256739
ft997	7.717563	4.600247	4.148705	8.741325	3.439116	7.895560	9.839678	6.953516	8.726596	10.436378
ft998	9.940031	7.003018	4.384623	6.155844	5.777876	9.105205	5.565315	8.056425	7.251609	7.126520
ft999	6.246147	5.142755	9.358403	7.194748	7.299672	7.907734	6.541831	6.999751	5.588490	6.211506
ft1000	5.752069	6.369917	5.263184	7.903122	10.687536	6.279037	8.859394	6.933735	11.505049	9.064493

Check the content of the data table from the second group.

```
dim(group2)
```

```
[1] 1000   16
```

```
kable(head(group2))
```

	g2s1	g2s2	g2s3	g2s4	g2s5	g2s6	g2s7	g2s8	g2s9	g2
ft1	9.455285	10.091510	8.268633	12.224986	10.687128	6.368127	11.637946	12.022067	14.546818	10.972
ft2	9.342032	10.966237	4.446881	5.930485	10.895613	10.649150	14.547814	8.589400	7.243914	2.645
ft3	12.892628	8.766024	12.413279	11.005271	8.764129	11.795945	9.745638	7.445941	10.943776	11.008
ft4	12.029128	11.836023	12.638985	11.703603	14.873479	5.462433	9.123551	9.547199	16.654958	8.675
ft5	9.781084	5.344621	11.854536	12.634565	11.179219	9.660712	12.125644	12.161726	10.979344	6.024
ft6	6.771381	9.119538	7.950780	11.574936	8.358685	8.064554	13.151311	14.297038	9.993751	8.786

```
kable(tail(group2))
```

	g2s1	g2s2	g2s3	g2s4	g2s5	g2s6	g2s7	g2s8	g2s9	
ft995	10.835650	13.776081	5.300087	6.406648	12.333749	4.721953	10.960206	14.857688	11.81474	14.2
ft996	14.316503	10.805613	11.357898	15.213863	9.646302	10.324418	6.676758	6.555227	10.69196	14.1
ft997	14.546172	9.575823	16.485424	9.625174	7.848070	10.193540	4.555923	4.894047	10.58934	11.4
ft998	13.779462	11.135147	9.851663	11.261584	14.371942	8.584143	15.202907	11.582024	11.10880	11.7
ft999	7.060473	6.320551	8.641556	15.720532	5.570717	12.092325	11.361147	9.709675	6.68905	11.9
ft1000	9.575039	14.191026	13.051048	10.349466	11.102488	13.106652	9.459714	8.161257	18.60538	9.4

Checking the properties of the data tables

Check the properties of the columns (individuals, e.g. biological samples) and rows (features, e.g. genes or proteins or metabolites) of the data tables.

- Use the `summary()` funciton to quickly inspect the column-wise properties (statistics per individual).
- Use `apply()`, `mean()` and `sd()` to generate a data frame that collects
  - the column-wise parameters (statistics per feature)
  - the row-wise parameters (statistics per feature).

```
## Column-wise summaries
summary(group1)
```

```
      g1s1             g1s2               g1s3               g1s4              g1s5               g1s6
 Min.   : 1.182   Min.   :-0.06562   Min.   :-0.09699   Min.   :-1.404   Min.   : 0.9575   Min.   :-1.5
 1st Qu.: 5.682   1st Qu.: 5.66581   1st Qu.: 5.57984   1st Qu.: 5.687   1st Qu.: 5.7425   1st Qu.: 5.6
 Median : 7.067   Median : 7.10060   Median : 6.93833   Median : 6.989   Median : 7.0133   Median : 7.0
 Mean   : 7.044   Mean   : 7.01922   Mean   : 6.96084   Mean   : 6.987   Mean   : 7.0043   Mean   : 7.0
 3rd Qu.: 8.371   3rd Qu.: 8.31442   3rd Qu.: 8.26930   3rd Qu.: 8.250   3rd Qu.: 8.3027   3rd Qu.: 8.4
 Max.   :13.443   Max.   :13.25858   Max.   :14.57354   Max.   :11.977   Max.   :14.5434   Max.   :13.6
```

```
summary(group2)
```

```
      g2s1              g2s2               g2s3              g2s4              g2s5               g2s6
 Min.   : 0.6336   Min.   :-0.09359   Min.   : 1.061   Min.   :-1.634   Min.   : 0.9045   Min.   :-0.62
 1st Qu.: 7.7375   1st Qu.: 8.11427   1st Qu.: 7.816   1st Qu.: 7.987   1st Qu.: 8.1487   1st Qu.: 7.67
 Median : 9.5952   Median :10.13660   Median : 9.867   Median : 9.849   Median :10.2131   Median : 9.99
 Mean   : 9.6988   Mean   :10.22668   Mean   : 9.878   Mean   : 9.890   Mean   :10.1951   Mean   : 9.88
 3rd Qu.:11.8660   3rd Qu.:12.16970   3rd Qu.:11.881   3rd Qu.:11.753   3rd Qu.:12.3358   3rd Qu.:12.10
 Max.   :17.2122   Max.   :20.69764   Max.   :19.044   Max.   :18.727   Max.   :18.8763   Max.   :18.43
```

```
## Columns-wise statistics
colStats <- data.frame(
    m1 = apply(group1, MARGIN = 2, mean),
    m2 = apply(group2, MARGIN = 2, mean),
    s1 = apply(group1, MARGIN = 2, sd),
    s2 = apply(group2, MARGIN = 2, sd)
  )

## Row-wise statistics
rowStats <- data.frame(
    m1 = apply(group1, MARGIN = 1, mean),
    m2 = apply(group2, MARGIN = 1, mean),
    s1 = apply(group1, MARGIN = 1, sd),
    s2 = apply(group2, MARGIN = 1, sd)
  )
```

Add a column with the difference between sample means for each feature.

Tips: this can be done in a single operation.

```
rowStats$diff <- rowStats$m1 - rowStats$m2
```

Merge the two groups in a single data frame

In omics data analysis, we typically obtain

- a single data table with all the individuals (biological samples) of all the groups
- a table containing the metadata, i.e. the information about each individual (biological sample)

Two methods could be envisaged a priori:

- `cbind()`, which simply binds the columns of two input tables. This can however be somewhat dangerous, because it assumes that these two tables have the same number of rows (features) and that these rows contain information about the same features *in the same order*. However, in some cases we dispose of data tables coming from different sources (or software tools), where the features (genes, proteins, metabolites) might have a partial overlap rather than an exact 1-to-1 correspondence, and, even when the feature sets are the same, they might be presented in different orderes.

- A much safer approach is thus to use `merge()`, and to explicitly indicate one or several colummns on which the features from the two table will be unified

In our case, the two data tables only contain numeric data, and the identification will be done on the row names (which contain the feature identifiers ft1, ft2, … that we defined above). In some cases, we will have to merge data table containing different informations, including a column with identifiers (or maybe additional columns e.g. the genotype, conditions, …) and we will use internal columns of the table to unify their rows.

We will create such a structure for further analysis/

```
## Read the help of the merge() function
# ?merge

## Create a data frame with the merged values
dataTable <- merge(x = group1, y = group2, by = "row.names")
# dim(dataTable) # NOTE: the merged table contains n1 + n2 columns + one additional column Row.names

## Use the Row.names column as names for the merged table
row.names(dataTable) <- dataTable$Row.names
dataTable <- dataTable[, -1] ## Suppress the first column which contained the row names
# dim(dataTable)
```

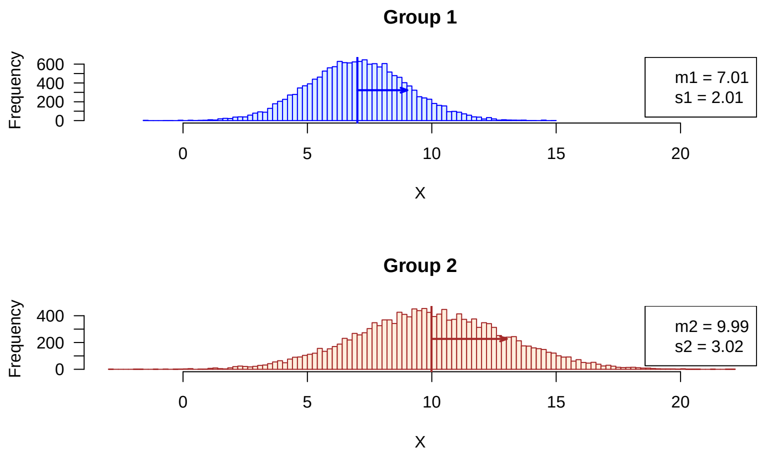
Visualisation of the data

- Draw two histograms with all the values group 1 and group2, respectively.

Tip: use `mfrow()` to display the histogram above each other, and set the limits of the abscissa (x axis) to the same value.

```
xlim  <- range(append(group1, group2))

## Compute mean and sd for all the samples of group1 and group2, resp
m1 <- mean(unlist(group1))
m2 <- mean(unlist(group2))
s1 <- sd(unlist(group1))
s2 <- sd(unlist(group2))

par(mfrow = c(2,1))
## Plot histogram of Group1 values, and get the values in a variable named h1
h1 <- hist(x = unlist(group1), breaks = 100, col = "#DDEEFF", border = "blue",
     main = "Group 1", xlab = "X", las = 1, xlim = xlim)
abline(v = m1, col = "blue", lwd = 2) ## mark the mean of all samples
arrows(x0 = m1, x1 = m1 + s1,
       y0 = max(h1$counts)/2, y1 = max(h1$counts)/2,
       length = 0.07, angle = 20, col = "blue", lwd = 2, code = 2)
```

```r
legend("topright",
       legend = c(paste0("m1 = ", round(digits = 2, m1)),
                  paste0("s1 = ", round(digits = 2, s1))))

## Plot histogram of Group2 values, and get the values in a variable named h2
h2 <- hist(x = unlist(group2), breaks = 100, col = "#FFEEDD", border = "brown",
     main = "Group 2", xlab = "X", las = 1, xlim = xlim)
abline(v = mean(unlist(group2)), col = "brown", lwd = 2)
arrows(x0 = m2, x1 = m2 + s2,
       y0 = max(h2$counts)/2, y1 = max(h2$counts)/2,
       length = 0.07, angle = 20, col = "brown", lwd = 2, code = 2)
legend("topright",
       legend = c(paste0("m2 = ", round(digits = 2, m2)),
                  paste0("s2 = ", round(digits = 2, s2))))
```



```r
par(mfrow = c(1,1))
```

- Draw histogram with the sampling distribution of the means in the respective groups.

```r
# xlim  <- range(append(colStats$m1, colStats$m2))

## Compute the parameters (mean, sd) of the sampling distributions of the means
mm1 <- mean(rowStats$m1)
mm2 <- mean(rowStats$m2)
se1 <- sd(rowStats$m1)
se2 <- sd(rowStats$m2)

par(mfrow = c(2,1))
## Plot histogram of Group1 values, and get the values in a variable named h1
h1 <- hist(x = rowStats$m1, breaks = 100, col = "#AACCFF", border = "#AACCFF",
     main = "Group 1", xlab = "X", las = 1, xlim = xlim)
abline(v = mm1, col = "blue", lwd = 2) ## mark the mean of all samples
arrows(x0 = mm1, x1 = mm1 + se1,
       y0 = max(h1$counts)/2, y1 = max(h1$counts)/2,
       length = 0.07, angle = 20, col = "blue", lwd = 2, code = 2)
legend("topright",
       legend = c(paste0("mm1 = ", round(digits = 2, mm1)),
                  paste0("se1 = ", round(digits = 2, se1))))

## Plot histogram of Group2 values, and get the values in a variable named h2
```

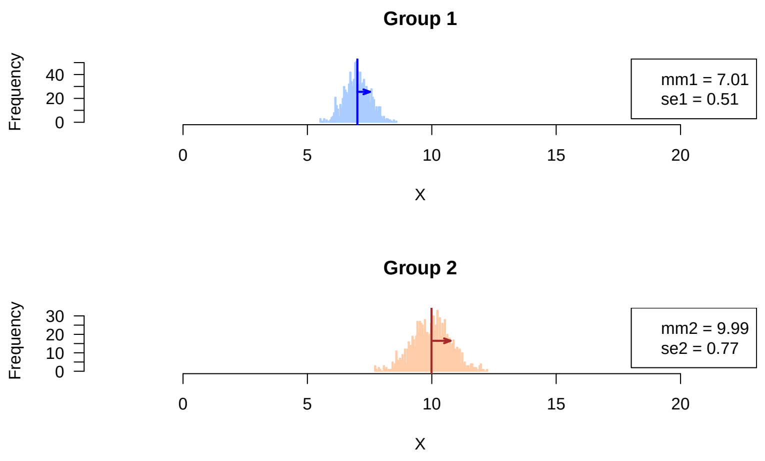


Figure 1: Sampling distribution of the mean

```
h2 <- hist(x = rowStats$m2, breaks = 100, col = "#FFCCAA", border = "#FFCCAA",
     main = "Group 2", xlab = "X", las = 1, xlim = xlim)
abline(v = mean(unlist(group2)), col = "brown", lwd = 2)
arrows(x0 = mm2, x1 = mm2 + se2,
       y0 = max(h2$counts)/2, y1 = max(h2$counts)/2,
       length = 0.07, angle = 20, col = "brown", lwd = 2, code = 2)
legend("topright",
       legend = c(paste0("mm2 = ", round(digits = 2, mm2)),
                  paste0("se2 = ", round(digits = 2, se2))))
```

```
par(mfrow = c(1,1))
```

- Compare the standard deviations measures in the sampled values, and in the feature means. Do they differ ? Explain why.

Answer: the standard deviation of the sample means corresponds to the **standard error**.

Part 2: hypothesis testing

Run Student test on a given feature

Since we are interested by differences in either directions, we run a two-tailed test.

Hypotheses:

$$H_0 : \mu_1 = \mu 2$$

$$H_1 : \mu_1 \neq \mu 2$$

Exercise: pick up a given feature (e.g. the 267^{th}) and run a mean comparison test. Choose the parameters according to your experimental setting.

Tips:

- `t.test()`
- you need to choose the test depending on whether the two populations have equal variance (Student) or not (Welch). Since we defined different values for the populations standard deviations (σ_1, σ_2), the choise is obvious.

```r
i <- 267 ## Pick up a given feature, arbitrarily

## Select the values for this feature in the group 1 and group 2, resp.
## Tip: I use unlist() to convert a single-row data.frame into a vector
x1 <- unlist(group1[i, ])
x2 <- unlist(group2[i, ])

## Run Sudent t test on one pair of samples
t.result <- t.test(
  x = x1, y = x2,
  alternative = "two.sided", var.equal = FALSE)

## Print the result of the t test
print(t.result)
```

```
    Welch Two Sample t-test

data:  x1 and x2
t = -2.9986, df = 22.277, p-value = 0.006557
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -5.1396591 -0.9387928
sample estimates:
mean of x mean of y
 5.595056  8.634282
```

```r
## Compute some additional statistics about the samples
mean1 <- mean(x1) ## Mean of sample 1
mean2 <- mean(x2) ## Mean of sample 2
d <- mean2 - mean1 ## Difference between sample means
```

Interpret the result

The difference between sample means was $d = 3.04$.

The t test computed the t statistics, which standardizes this observed distance between sample means relative to the estimated variance of the population, and to the sample sizes. With the random numbers generated above, the value is $t_{obs} = -2.9986$.

The corresponding p-value is computed as the sum of the area of the left and right tails of the Student distribution, with $\nu = n_1 + n_2 - 2 = 22.2772204$ degrees of freedom. It indicates the probability of obtaining by chance – ***under the null hypothesis*** – a result at least as extreme as the one we observed.

In our case, we obtain $p = P(T > |t_{obs}| = P(T > 2.9986) = 0.00656$. This is higher than our threshold $alpha = 0.05$. We thus accept the null hypothesis.

Replicating the test for each feature

In R, loops are quite inefficient, and it is generally recommended to directly run the computations on whole vectors (R has been designed to be efficient for this), or to use specific functions in order to apply a given function each row / column of a table, or to each element of a list.

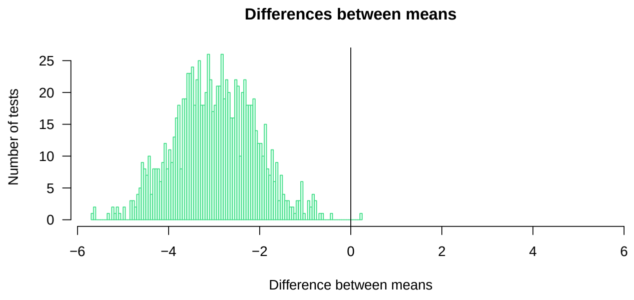
For the sake of simplicity, we will first show how to implement a simple but inefficient code with a loop. In the advanced course (STATS2) will see how to optimize the speed with the `apply()` function.

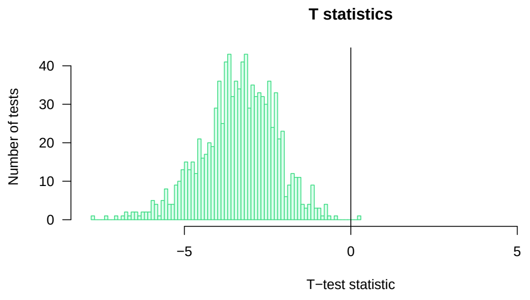
```r
## Define the statistics we want to collect
resultColumns <- c("i",       # iteration number
                    "m1",      # first sample mean
                    "m2",      # second sample mean
                    "s1",      # sd estimation for the first population
                    "s2",      # sd estimation for the second population
                    "d",       # difference between sample means
                    "t",       # test statistics
                    "df",      # degrees of freedom
                    "p.value"  # nominal p-value
                    )

## Instantiate a result table to store the results
resultTable <- data.frame(matrix(nrow = m, ncol = length(resultColumns)))
colnames(resultTable) <- resultColumns # set the column names
# View(resultTable) ## Check the table: it contians NA values

## Iterate random number sampling followed by t-tests
## Use the funciton system.time() to measure the elapsed time
## This function is particular: you can also use it with curly brackets in order to enclose a block of 
time.iteration <- system.time(
  for (i in 1:m) {
    ## Generate two vectors containing the values for sample 1 and sample 2, resp.
    x1 <- unlist(group1[i, ]) ## sample 1 values
    x2 <- unlist(group2[i, ]) ## sample 2 values

    ## Run the t test
    t.result <- t.test(
      x = x1, y = x2,
      alternative = "two.sided", var.equal = FALSE)
    # names(t.result)

    ## Collect the selected statistics in the result table
    resultTable[i, "i"] <- i
    resultTable[i, "t"] <- t.result$statistic
    resultTable[i, "df"] <- t.result$parameter
    resultTable[i, "p.value"] <- t.result$p.value

    ## Compute some additional statistics about the samples
    resultTable[i, "m1"] <- mean(x1) ## Mean of sample 1
    resultTable[i, "m2"] <- mean(x2) ## Mean of sample 2
    resultTable[i, "s1"] <- sd(x1) ## Standard dev estimation for population 1
    resultTable[i, "s2"] <- sd(x2) ## Standard dev estimation for population 1
    resultTable[i, "d"] <- resultTable[i, "m1"] - resultTable[i, "m2"]  ## Difference between sample me
  }
  #}
  ## View(resultTable)
)

print(time.iteration)
```

```
   user  system elapsed 
  0.581   0.014   0.597 
```

Distribution of the observed differences for the 1000 iterations of the test

```
par(mfrow = c(2, 1))
#head(resultTable)

## Compute the maximal abslute value of difference to get a centered abcsissa.
## This enables to highlight whether the differences are positive or negative.
max.diff <- max(abs(resultTable$d))

## Draw an histogram of the observed differences
hist(resultTable$d,
     breaks = 100,
     col = "#DDFFEE",
     border = "#44DD88",
     las = 1,
     xlim = c(-max.diff, max.diff), ## Make sure that the graph is centered on 0
     main = "Differences between means",
     xlab = "Difference between means",
     ylab = "Number of tests")
abline(v = 0)

max.t <- max(abs(resultTable$t))
hist(resultTable$t,
     breaks = 100,
     col = "#DDFFEE",
     border = "#44DD88",
          las = 1,
     xlim = c(-max.t, max.t), ## Make sure that the graph is centered on 0
     main = "T statistics",
     xlab = "T-test statistic",
     ylab = "Number of tests")
abline(v = 0)
```

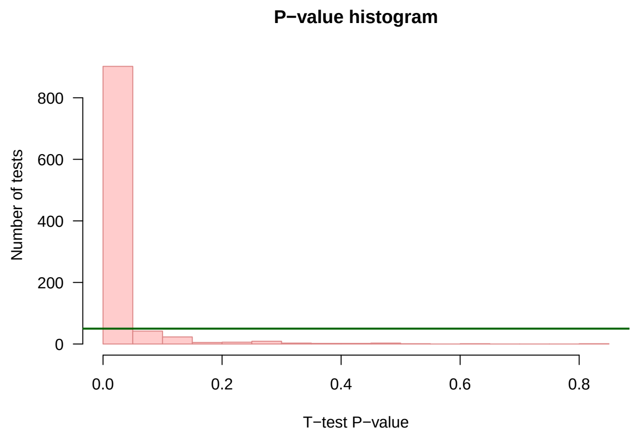




```
par(mfrow = c(1, 1))
```

P-value histogram

```
## Choose a color depending on whether we are under H0 (grey) or H1 (pink)
if (mu1 == mu2) {
  histColor <- "#DDDDDD"
  histBorder <- "#888888"
} else {
  histColor <- "#FFCCCC"
  histBorder <- "#DD8888"
}

## Draw an histogram of p-values with 20 bins
hist(resultTable$p.value,
     breaks = 20,
     col = histColor,
     border = histBorder,
     las = 1,
     main = "P-value histogram",
     xlab = "T-test P-value",
     ylab = "Number of tests")

## Draw a horizontal line indicating the number of tests per bin that would be expected under null hypo
abline(h = m / 20, col = "darkgreen", lwd = 2)
```



Creating a function to reuse the same code with different parameters

Depending on the selected task in the assignments above, we will run different tests with different parameters and compare the results. The most rudimentary way to do this is top copy-paste the chunk of code above for each test and set of parameters required for the assigned tasks.

However, having several copies of an almost identical block of code is a very bad pracrice in programming, for several reasons

- lack of readability: the code rapidly becomes very heavy;
- difficulty to maintain: any modification has to be done on each copy of the chunk of code;
- risk for consistency: this is a source of inconsistency, because at some moment we will modify one copy and forget another one.

A better practice is to define a **function** that encapsulates the code, and enables to modify the parameters by passing them as **arguments*. Hereafter we define a function that

- takes the parameters of the analysis as arguments
  - population means μ_1 and μ_2,
  - population standard deviations σ_1 and σ_2,
  - sample sizes n_1 and n_2,
  - number of iterations r.
- runs r iterations of the t-test with 2 random samples,
- returns the results in a table with one row per iteration, and one column per resulting statistics (observed t score, p-value, difference between means, ...);

```
#### Define a function that runs r iterations of the t-test ####

#' @title Repeat a T test with random numbers drawn from a normal distribution
#' @param mu1 mean of the first population
#' @param mu2 mean of the second population
#' @param sigma1 standard deviation of the first population
#' @param sigma2 standard deviation of the second population
#' @param n1 first sample size
#' @param n2 second sample size
#' @param m number of repetitions of the tests
#' @param ... additional parameters are passed to t.test(). This enables to set var.equal, alternative,
#' @return  a table with one row per repetition of the test, and one column per statistics
IterateTtest <- function(mu1,
                         mu2,
                         sigma1,
                         sigma2,
                         n1,
                         n2,
                         m,
                         ...) {


  ## Define the statistics we want to collect
  resultColumns <- c("i",       # iteration number
                     "m1",      # first sample mean
                     "m2",      # second sample mean
                     "s1",      # sd estimation for the first population
                     "s2",      # sd estimation for the second population
                     "d",       # difference between sample means
                     "t",       # test statistics
                     "df",      # degrees of freedom
                     "p.value"  # nominal p-value
  )

  ## Instantiate a result table to store the results
  resultTable <- data.frame(matrix(nrow = m, ncol = length(resultColumns)))
  colnames(resultTable) <- resultColumns # set the column names
  # View(resultTable) ## Check the table: it contians NA values

  ## Iterate random number sampling followed by t-tests
  for (i in 1:m) {
```

```r
    ## Generate two vectors containing the values for sample 1 and sample 2, resp.
    x1 <- rnorm(n = n1, mean = mu1, sd = sigma1) ## sample 1 values
    x2 <- rnorm(n = n2, mean = mu2, sd = sigma2) ## sample 2 values

    ## Run the t test
    t.result <- t.test(
      x = x1, y = x2,
      alternative = "two.sided", var.equal = TRUE)
    # names(t.result)

    ## Collect the selected statistics in the result table
    resultTable[i, "i"] <- i
    resultTable[i, "t"] <- t.result$statistic
    resultTable[i, "df"] <- t.result$parameter
    resultTable[i, "p.value"] <- t.result$p.value

    ## Compute some additional statistics about the samples
    resultTable[i, "m1"] <- mean(x1) ## Mean of sample 1
    resultTable[i, "m2"] <- mean(x2) ## Mean of sample 2
    resultTable[i, "s1"] <- sd(x1) ## sd estimate for population 1
    resultTable[i, "s2"] <- sd(x2) ## sd estimate for population 2
    resultTable[i, "d"] <- resultTable[i, "m1"] - resultTable[i, "m2"] ## Difference between sample mea
  }

  return(resultTable) ## This function returns the result table
}
```

We can now use this function to iterate the t test with the parameters we want. Let us measure the running time

```r
## Some tests under H1
system.time(
  tTestTableH1 <- IterateTtest(mu1 = 7, mu2 = 10, sigma1 = 2, sigma2 = 3, n1 = 16, n2 = 16, m = 1000)
)
```

```
   user  system elapsed 
  0.348   0.006   0.355 
```

```r
## Some tests under H0
system.time(
  tTestTableH0 <- IterateTtest(mu1 = 10, mu2 = 10, sigma1 = 2, sigma2 = 3, n1 = 16, n2 = 16, m = 1000)
)
```

```
   user  system elapsed 
  0.335   0.007   0.343 
```

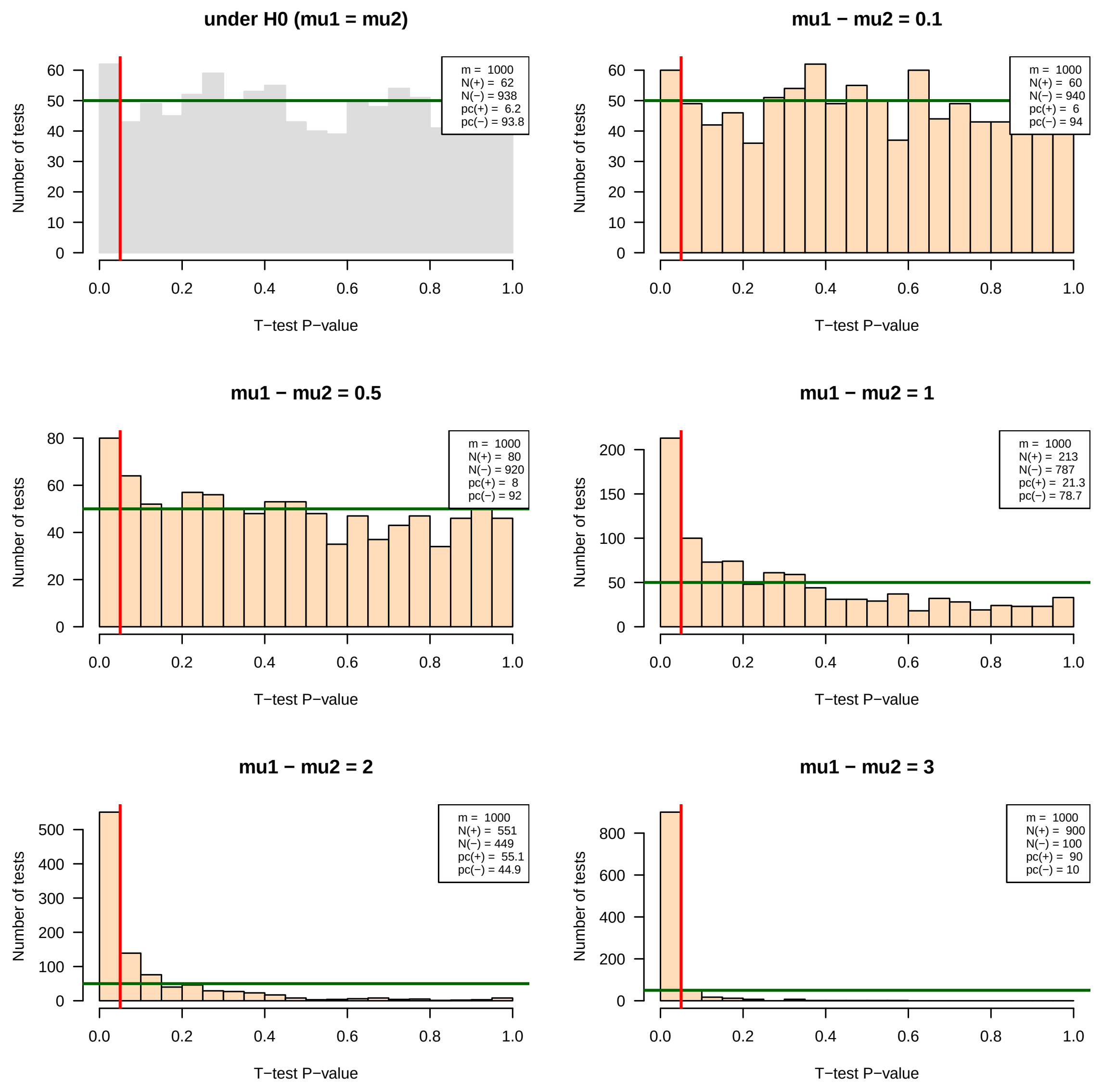
This function can then be used several times, with different values of the parameters.

```r
## What happens when  the two means are equal (under the null hypothesis)
testH0 <- IterateTtest(mu1 = 10, mu2 = 10, sigma1 = sigma1, sigma2 = sigma2, n1 = n1, n2 = n2, m = m, va


## Test increasing values of the difference between population means (delta)
delta0.1 <- IterateTtest(mu1 = mu1, mu2 = mu1 + 0.1, sigma1 = sigma1, sigma2 = sigma2, n1 = n1, n2 = n2

delta0.5 <- IterateTtest(mu1 = mu1, mu2 = mu1 + 0.5, sigma1 = sigma1, sigma2 = sigma2, n1 = n1, n2 = n2

delta1 <- IterateTtest(mu1 = mu1, mu2 = mu1 + 1, sigma1 = sigma1, sigma2 = sigma2, n1 = n1, n2 = n2, m =

delta2 <- IterateTtest(mu1 = mu1, mu2 = mu1 + 2, sigma1 = sigma1, sigma2 = sigma2, n1 = n1, n2 = n2, m =

delta3 <- IterateTtest(mu1 = mu1, mu2 = mu1 + 3, sigma1 = sigma1, sigma2 = sigma2, n1 = n1, n2 = n2, m =
```

```
## Define a function that rdraws the p-value histogram
## based on the result table of t-test iterations
## as produced by the iterate.t.test() function.
pvalHistogram <- function(
  resultTable, ## required input (no default value): the result table from iterate.t.test()
  main = "P-value histogram",  ## main title (with default value)
  alpha = 0.05, ## Significance threshold
  ... ## Additional parameters, which will be passed to hist()
  ) {

  ## Plot the histogram
  hist(resultTable$p.value,
       breaks = seq(from = 0, to = 1, by = 0.05),
       las = 1,
       xlim = c(0,1),
       main = main,
       xlab = "T-test P-value",
       ylab = "Number of tests",
       ...)

  ## Draw a horizontal line indicating the number of tests per bin that would be expected under null hyp
  abline(h = m / 20, col = "darkgreen", lwd = 2)
  abline(v = alpha, col = "red", lwd = 2)

  ## Compute the percent of positive and negative results``
  nb.pos <- sum(resultTable$p.value < alpha)
  nb.neg <- m - nb.pos
  percent.pos <- 100 * nb.pos / m
  percent.neg <- 100 * nb.neg / m

  ## Add a legend indicating the percent of iterations declaed positive and negative, resp.
  legend("topright",
         bty = "o",
         bg = "white",
         cex = 0.7,
         legend = c(
           paste("m = ", nrow(resultTable)),
           paste("N(+) = ", nb.pos),
           paste("N(-) =", nb.neg),
           paste("pc(+) = ", round(digits = 2, percent.pos)),
           paste("pc(-) =", round(digits = 2, percent.neg))
         ))
}
```

```
par(mfrow = c(3, 2)) ## Prepare 2 x 2 panels figure
pvalHistogram(testH0, main = "under H0 (mu1 = mu2)", col = "#DDDDDD", border = "#DDDDDD")
pvalHistogram(delta0.1, main = "mu1 - mu2 = 0.1", col = "#FFDDBB")
pvalHistogram(delta0.5, main = "mu1 - mu2 = 0.5", col = "#FFDDBB")
```

```r
pvalHistogram(delta1, main = "mu1 - mu2 = 1", col = "#FFDDBB")
pvalHistogram(delta2, main = "mu1 - mu2 = 2", col = "#FFDDBB")
pvalHistogram(delta3, main = "mu1 - mu2 = 3", col = "#FFDDBB")
```



```r
par(mfrow = c(1, 1)) ## Restore single-panel layout for next figures
```

Interpretation of the results

We should now write a report of interpretation, which will address the following questions.

- Based on the experiments under H_0, compute the number of false positives and estimate the **false positive rate** (**FPR**). Compare these values with the **E-value** (expected number of false positives) for the 1000 tests, and with your *alpha* treshold.
- Based on the experiments under H_1, estimate the **sensitivity** (**Sn**) of the test for the different mean differences tested here.
- Interpret the histograms of P-values obtained with the different parameters ?
- Draw a **power curve** (i.e. the sensitivity as a function of the actual difference between population

means)

- Discuss about the adequation between the test and the conditions of our simulations.
- Do these observations correspond to what would be expected?

The same kind of questions will be asked for the 6 other questions above (impact of sample size, variance, non-normality, heteroscadicity, parametric vs non-parametric test).