# Tutorial – Comparing means under controlled conditions

Diplôme Universitaire en Bioinformatique Intégrative (DUBii)

*Jacques van Helden*

*2020-03-11*

## 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 |
| $\mu_1$ | 10 or 7 | mean of the first population. each participant will chose one value |
| $\mu_2$ | 10 | mean of the second population |
| $\sigma_1$ | 2 | Standard deviation of the first population |
| $\sigma_2$ | 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 → 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 <- 8 # sample size for the first group
n2 <- 8 # ssample size for second group

## First data table
m <- 10000 # 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 |
|---|---|---|
| $\mu_1$ | 7 | Mean of the first population |
| $\mu_2$ | 10 | Mean of the second population |
| $\sigma_1$ | 2 | Standard deviation of the first population |
| $\sigma_2$ | 2 | Standard deviation of the second population |
| $n_1$ | 8 | Sample size for the first group |
| $n_2$ | 8 | 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: `grp1s1`, , `grp1s2` ... with indices from 1 $n_1$.
- Name the rows to denote the feature numbers: `h0ft1`, `h0ft2`, ... 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).

```
#### Generating a random data frame of the appropriate size ####

## Dataset under H0
group1 <- data.frame(
  matrix(data = rnorm(n = m * n1, mean = mu1, sd = sigma1),
         nrow = m,
         ncol = n1))
colnames(group1) <- paste(sep = "", "grp1s", 1:n1)
rownames(group1) <- paste(sep = "", "h0ft", 1:m)

## Dataset under h1
group2 <- data.frame(
  matrix(data = rnorm(n = m * n2, mean = mu2, sd = sigma2),
         nrow = m,
         ncol = n2))
colnames(group2) <- paste(sep = "", "grp2s", 1:n2)
rownames(group2) <- paste(sep = "", "h0ft", 1:m)
```

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

```
dim(group1)
```

```
[1] 10000     8
```

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

| | grp1s1 | grp1s2 | grp1s3 | grp1s4 | grp1s5 | grp1s6 | grp1s7 | grp1s8 |
|---|---|---|---|---|---|---|---|---|
| h0ft1 | 6.261325 | 5.720199 | 11.038376 | 6.422049 | 6.270067 | 7.766138 | 9.137222 | 9.952895 |
| h0ft2 | 6.487359 | 10.388523 | 7.626327 | 7.340299 | 5.716033 | 10.220082 | 6.389231 | 9.853454 |
| h0ft3 | 10.154002 | 6.618097 | 7.120213 | 7.102505 | 8.104277 | 6.529419 | 8.381264 | 3.350799 |
| h0ft4 | 7.682733 | 7.652766 | 8.345135 | 7.376686 | 8.623861 | 7.513706 | 4.280821 | 5.197220 |
| h0ft5 | 7.340649 | 8.538659 | 7.965929 | 6.245233 | 9.802664 | 7.016332 | 7.513553 | 9.228668 |
| h0ft6 | 7.751690 | 10.317747 | 6.754905 | 6.559597 | 5.299607 | 3.789787 | 6.685665 | 6.301232 |

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

| | grp1s1 | grp1s2 | grp1s3 | grp1s4 | grp1s5 | grp1s6 | grp1s7 | grp1s8 |
|---|---|---|---|---|---|---|---|---|
| h0ft9995 | 5.336209 | 5.789849 | 14.826660 | 8.902123 | 5.038242 | 7.368357 | 5.697458 | 7.379172 |
| h0ft9996 | 10.395149 | 4.668423 | 3.998208 | 5.869831 | 8.221302 | 6.307620 | 3.684371 | 6.095613 |
| h0ft9997 | 5.768135 | 5.262576 | 6.531441 | 7.857153 | 6.049440 | 5.701706 | 5.917838 | 7.110208 |
| h0ft9998 | 7.677099 | 4.537946 | 8.735657 | 6.614589 | 6.542243 | 11.754664 | 5.070481 | 6.234585 |
| h0ft9999 | 5.852029 | 9.787215 | 6.188260 | 6.123055 | 8.428192 | 5.307171 | 8.879681 | 6.205750 |
| h0ft10000 | 2.679708 | 6.273860 | 4.745285 | 7.152692 | 7.453945 | 7.911892 | 7.767660 | 8.365867 |

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

```
dim(group2)
```

```
[1] 10000     8
```

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

| | grp2s1 | grp2s2 | grp2s3 | grp2s4 | grp2s5 | grp2s6 | grp2s7 | grp2s8 |
|---|---|---|---|---|---|---|---|---|
| h0ft1 | 3.065207 | 9.047130 | 7.049548 | 11.728788 | 8.544925 | 11.697647 | 13.290798 | 12.277071 |
| h0ft2 | 4.749248 | 7.291797 | 11.976171 | 11.581664 | 7.175354 | 8.742851 | 12.609519 | 6.966799 |
| h0ft3 | 1.868618 | 12.064263 | 14.010469 | 6.095285 | 11.188184 | 11.058078 | 9.422339 | 12.489879 |
| h0ft4 | 11.136922 | 11.541562 | 10.512444 | 13.014590 | 11.468524 | 8.146551 | 12.174131 | 8.038919 |
| h0ft5 | 7.716118 | 5.329844 | 10.941822 | 7.659905 | 13.559551 | 10.785339 | 9.617110 | 6.675426 |
| h0ft6 | 12.187237 | 11.925986 | 7.201090 | 6.648762 | 9.518069 | 10.932162 | 14.015160 | 12.022674 |

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

| | grp2s1 | grp2s2 | grp2s3 | grp2s4 | grp2s5 | grp2s6 | grp2s7 | grp2s8 |
|---|---|---|---|---|---|---|---|---|
| h0ft9995 | 13.364252 | 16.498548 | 10.92838 | 3.309989 | 8.136489 | 11.449191 | 4.726702 | 7.579818 |
| h0ft9996 | 9.152688 | 7.863256 | 11.70069 | 10.336449 | 8.304407 | 6.276297 | 8.948510 | 8.925372 |
| h0ft9997 | 9.249471 | 12.252379 | 11.10364 | 13.779583 | 9.410507 | 11.609783 | 9.826781 | 15.161252 |
| h0ft9998 | 13.130346 | 9.157739 | 13.59047 | 8.750145 | 11.397139 | 15.589427 | 9.424897 | 10.711849 |
| h0ft9999 | 10.485257 | 14.399796 | 10.01593 | 7.262640 | 10.102167 | 11.177083 | 15.915607 | 7.855550 |
| h0ft10000 | 10.362867 | 9.195259 | 15.98499 | 8.532756 | 5.941811 | 11.423079 | 6.781653 | 9.272903 |

### Checking the properties of the result table

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).

```
## Columns-wise statistics
colStatsH0 <- 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
rowStatsH0 <- 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.

```
rowStatsH0$diff <- rowStatsH0$m2 - rowStatsH0$m1
```

**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.

- Draw histogram with the sampling distribution of the means in the respective groups.
- Compare the standard deviations measures in the sampled values, and in the feature means. Do they differ ? Explain why.

## 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 ($\sigma_1$, $\sigma_2$), the choise is obvious.

```
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, ])
```

```r
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 = -3.5025, df = 11.679, p-value = 0.004535
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -8.364332 -1.936709
sample estimates:
mean of x mean of y
  6.98295  12.13347
```

```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 $d = 5.15$.

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 $t_{obs} = -3.5025$.

The corresponding p-value is computed as the sum of the area of the left and right tails of the Student distribution, with $\nu = n_1 + n_2 - 2 = 11.6793745$ 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 $p = P(T > |t_{obs}| = P(T > 3.5025) = 0.00454$. 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.

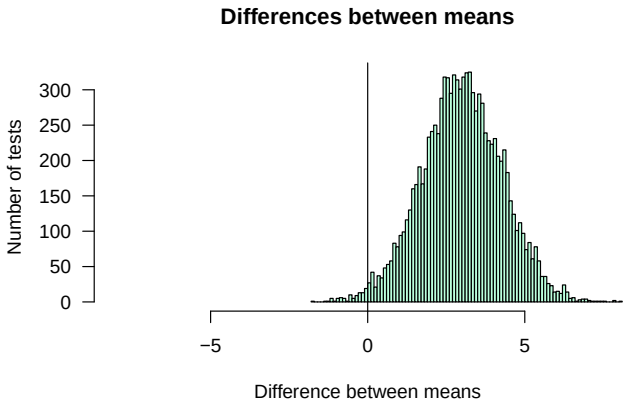
For the sake of simplicity, we will first show how to implement a simple but inefficient code with a loop. In the advanced course (STATS2) will see how to optimize the speed with the `apply()` function.

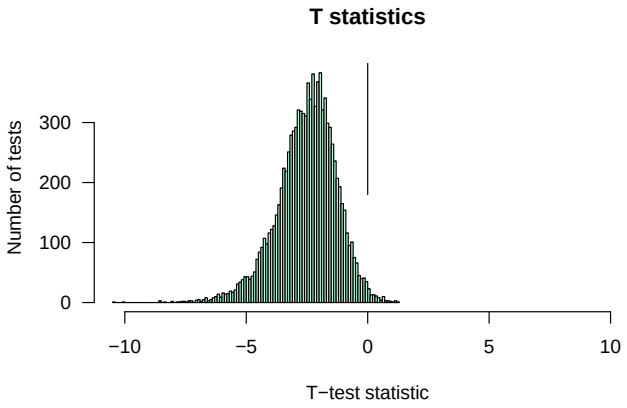
```r
## Define the statistics we want to collect
result.columns <- c("i", "m1", "m2", "diff", "statistic", "p.value")

## Instantiate a result table to store the results
result.table <- data.frame(matrix(nrow = m, ncol = length(result.columns)))
colnames(result.table) <- result.columns # set the column names
```

```r
# View(result.table) ## Check the table: it contians NA values

## Iterate random number sampling followed by t-tests
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
  result.table[i, "i"] <- i
  result.table[i, "statistic"] <- t.result["statistic"]
  result.table[i, "p.value"] <- t.result["p.value"]

  ## Compute some additional statistics about the samples
  result.table[i, "m1"] <- mean(x1) ## Mean of sample 1
  result.table[i, "m2"] <- mean(x2) ## Mean of sample 2
  result.table[i, "diff"] <- mean(x2) - mean(x1)  ## Difference between sample means

}

## View(result.table)
```

**Distribution of the observed differences for the $10^4$ iterations of the test**

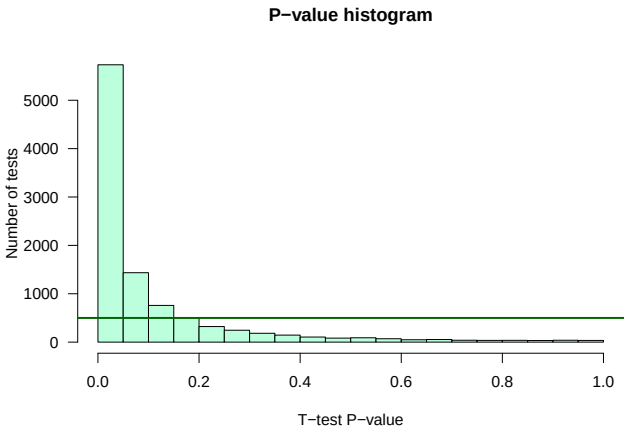
```r
par(mfrow = c(2, 1))
#head(result.table)
## Draw an histogram of the observed differences
max.diff <- max(abs(result.table$diff))
hist(result.table$diff, breaks = 100,
     col = "#BBFFDD", las = 1,
     xlim = c(-max.diff, max.diff), ## Make sure that the graph is centered on 0
     main = "Differences between means",
     xlab = "Difference between means",
     ylab = "Number of tests")
abline(v = 0)

max.t <- max(abs(result.table$statistic))
hist(result.table$statistic, breaks = 100,
     col = "#BBFFDD", 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")
par(mfrow = c(1, 1))
abline(v = 0)
```




### P-value histogram

```
## Draw an histogram of p-values with 20 bins
hist(result.table$p.value, breaks = 20,
     col = "#BBFFDD", 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 $\mu_1$ and $\mu_2$,
  - population standard deviations $\sigma_1$ and $\sigma_2$,
  - 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.
iterate.t.test <- function(mu1 = -0.5, ## Mean of the first population
                           mu2 = 0.5,  ## Mean of the second population
                           sigma1 = 1, ## Standard deviation of the first population
                           sigma2 = 1,  ## Standard deviation of the second population
                           n1 = 10,    ## First sample size
                           n2 = 10,    ## Second sample size
                           r = 10000  ## Iterations
                           ) {

  ## Define the statistics we want to collect
  result.columns <- c("i", "m1", "m2", "diff", "statistic", "p.value")

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

  ## Iterate random number sampling followed by t-tests
  for (i in 1: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
    result.table[i, "i"] <- i
    result.table[i, "statistic"] <- t.result["statistic"]
    result.table[i, "p.value"] <- t.result["p.value"]

    ## Compute some additional statistics about the samples
    result.table[i, "m1"] <- mean(x1) ## Mean of sample 1
    result.table[i, "m2"] <- mean(x2) ## Mean of sample 2
    result.table[i, "diff"] <- mean(x2) - mean(x1) ## Difference between sample means
  }

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

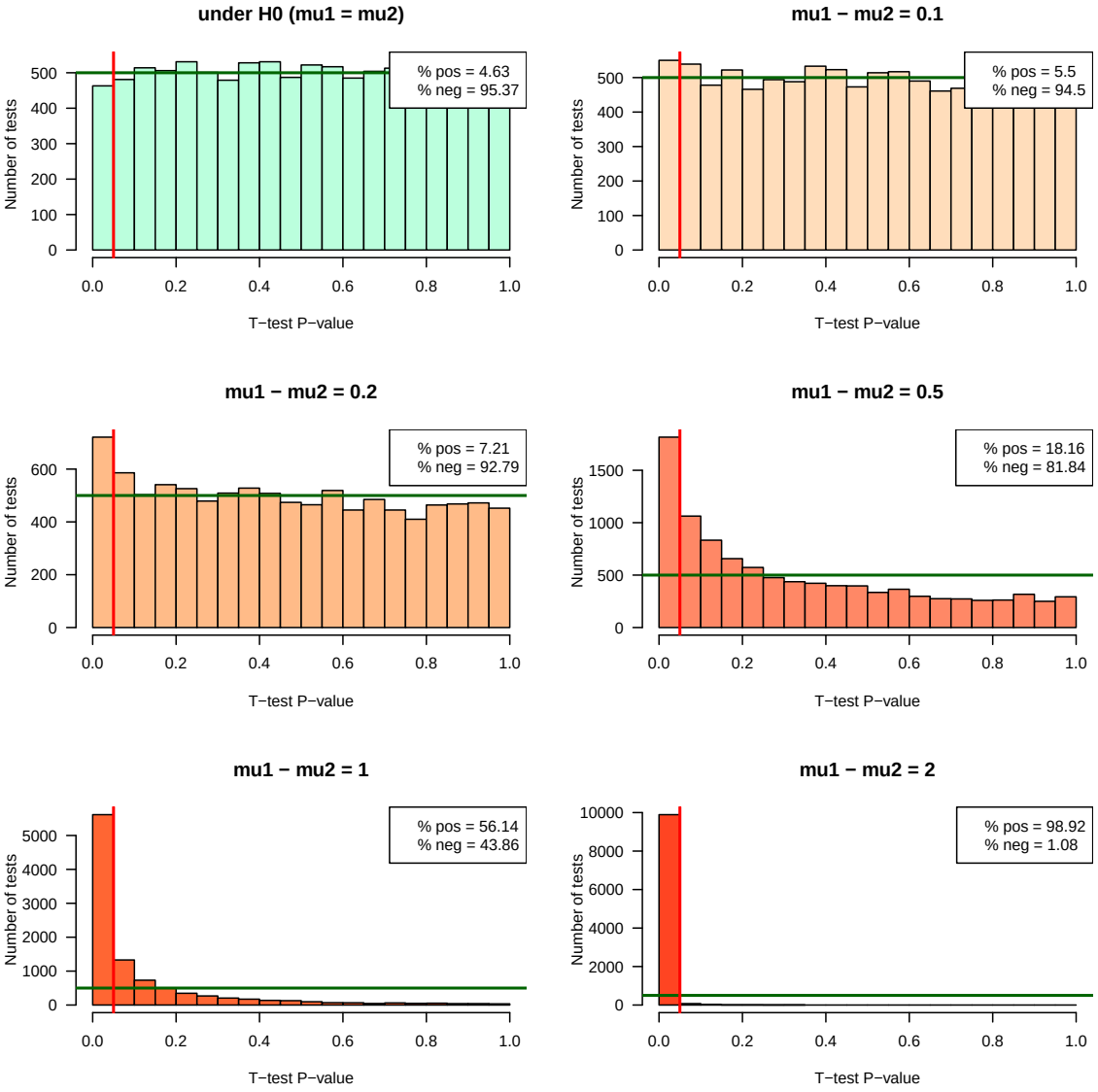
This function can then be used several times, with different values of the parameters.

```
## What happens when  the two means are equal (under the null hypothesis)
d0 <- iterate.t.test(mu1 = 0, mu2 = 0)  ## Iterations

## Test increasing values of the difference between means
d0.1 <- iterate.t.test(mu1 = -0.05, mu2 = 0.05)  ## Iterations
d0.2 <- iterate.t.test(mu1 = -0.1, mu2 = 0.1)  ## Iterations
d0.5 <- iterate.t.test(mu1 = -0.25, mu2 = 0.25)  ## Iterations
d1 <- iterate.t.test(mu1 = -0.5, mu2 = 0.5)  ## Iterations
d2 <- iterate.t.test(mu1 = -1, mu2 = 1)  ## Iterations
```

```
## 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(
  result.table, ## 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(result.table$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(result.table$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",
         legend = c(
           paste("% pos =", round(digits = 2, percent.pos)),
           paste("% neg =", round(digits = 2, percent.neg))
         ))
}
```

```
par(mfrow = c(3, 2)) ## Prepare 2 x 2 panels figure
pvalHistogram(d0, main = "under H0 (mu1 = mu2)", col = "#BBFFDD")
pvalHistogram(d0.1, main = "mu1 - mu2 = 0.1", col = "#FFDDBB")
pvalHistogram(d0.2, main = "mu1 - mu2 = 0.2", col = "#FFBB88")
pvalHistogram(d0.5, main = "mu1 - mu2 = 0.5", col = "#FF8866")
pvalHistogram(d1, main = "mu1 - mu2 = 1", col = "#FF6633")
pvalHistogram(d2, main = "mu1 - mu2 = 2", col = "#FF4422")
```



```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 $10^4$ tests, and with your *alpha* trheshold.
- 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).