GATTACA Manual

Visentin Luca and Ruffinatti A. Federico, Ph.D. - December 2021

Contents

1 Introduction . . . **3**
1.1 Installation . . . 4

2 General options . . . **4**

3 Data Preparation . . . **4**

4 Differential Gene Expression Analysis . . . **6**
4.1 Create a config file . . . 6
4.2 Customize the config file . . . 6
4.3 Running GATTACA . . . 8
4.4 Defining the experimental design . . . 8
4.4.1 Design strings . . . 8
4.4.2 The experimental design string . . . 9
4.4.3 Specifying technical replicates . . . 10
4.4.4 Specifying batches . . . 10
4.4.5 Additional limma variables . . . 10
4.5 Interpreting results . . . 10

5 Annotating Results . . . **11**

6 Quality control plots . . . **12**
6.1 MA plots . . . 12
6.2 Expression Boxplots . . . 13
6.3 Dendrogram, PCA and Scree plots . . . 13
6.4 Poisson plots . . . 15
6.5 Volcano Plots . . . 17

7 Computational reproducibility . . . **17**

8 Install and use GATTACA interactively . . . **19**
8.1 Contributing . . . 19

Bibliograpy . . . **19**

Appendices **20**

A Mini YAML guide . . . **20**

About the formatting of this document: `Monospace` text refers to a package, command, programming language or other programming-related concept. Text in green refers to an entry in the glossary. Text in this blue is a link to an external source. Text in this blue is a link to a section inside this document. Square brackets with violet numbers are citations.

This document covers `GATTACA` version `0.3.2`.

1 Introduction

A common wish for a researcher in the genomics field is to compare the gene expression[1] of two (or more) different samples and obtain a list of Differentially Expressed Genes (DEGs). This procedure is known as Differential Expression Analysis (DEA), and involves four steps: obtaining gene expression data, data preprocessing, running DEA, and annotation.

A - now almost obsolete - method of obtaining gene expression data is the usage of **microarrays**. Microarrays are solid chips upon which DNA probes are linked. mRNA is extracted from the samples, purified, amplified and retrotranscribed to cDNA. The resulting libraries are then applied on the chips. The probes will link to complementary sequences, and, after washing and using specific imaging methods, the amount of linkage for each probe can be detected. The resulting intensities are directly correlated with the expression of the mRNA in the samples.

Raw intensity data from microarry scanners (henceforth "raw data") needs to be preprocessed before DEGs can be detected. In the context of this document, we define "Microarray Data Preprocessing (MDP)" as the set of steps that need to be performed on such data to obtain an **expression dataset**. An expression dataset is a matrix with columns representing different samples (microarrays) and rows representing genes detected by the microarrays. The values in the matrix represent the `log2` expression values for that gene/sample combo. MDP is a multi-step, platform-dependent process. Many packages are available in the R programming language to perform MDP, for example `limma` [1], `oligo`, and `affycoretools`. Knowledge of R is needed to use these tools.

One of the most widely known and used tools to perform DEA is `limma`, which uses (generalized) linear models to detect DEGs. `RankProd` is another package that can detect DEGs using non-parametric, rank-based approaches [2]. Both these packages are implemented in the R language, and require some knowledge of R to be used.

Finally, microarray scanners will label the intensities with their respective probe IDs, while an human researcher prefers Gene Symbols (or Human Genome Organization (HUGO) symbols). This step is called *annotation*, and requires the use of one of many annotation databases that hold the correspondences between probe IDs and Gene Symbols, among other.

There is currently a need for a simple tool that can analyse microarray expression data to produce a list of DEGs using an intuitive, user friendly interface. We present General Algorithm for The Transcriptional Analysis by one-Channel Arrays (GATTACA), a self-contained tool to prepare, analyse and annotate microarray gene expression data. An additional benefit of GATTACA is its reproducibility. Being containerized, the tool is guaranteed to be computationally reproducible.

One of the strengths of GATTACA is its user friendliness, so that any researcher, even without a background in bioinformatics or knowledge of R, can correctly analyze both novel and pre-existing datasets.

[1] More formally, the amount of mRNA in the sample, which often correlates directly with the amount of expressed protein.

GATTACA is written in R, containerized in Docker and fully Open-Source. The source code is hosted on GitHub. Users are encouraged to propose enhancements and report issues on the website. GATTACA runs only on unix-like operating systems.

1.1 Installation

To install the tool, install `Docker` (following the official guides) and `curl` (with your distribution's package manager of choice). Move into the folder where you'd like to install the tool, then run:

```
curl https://raw.githubusercontent.com/Feat-FeAR/GATTACA/main/GATTACA --output GATTACA \
&& chmod +x ./GATTACA
```

Assuming that the folder where the script is placed is in your `PATH`, the tool can be called with the `GATTACA` command. This may be tested by running `GATTACA --help`. If the command resolves with no errors, the tool is installed correctly.

2 General options

GATTACA can be called in the following way:

```
GATTACA [-h | --help] [-v | --version <version>] [-l | --log_name <name>] <command> [<args>]
```

- The `-h | --help` flag ends the command early and prints a help message;
- The `-v | --version <version>` optional parameter specifies the version of the docker to run. To see a list of the available containers, refer to the Github release page. If unset, the `latest` docker release is used instead.
- The `-l | --log_name <name>` parameter specifies the filename of the log file that is always created in the output folder. If unset, the filename is generated with the current date and time, and is therefore unique for each run.

The following sections describe the possible `command`s and their respective `args`.

3 Data Preparation

To prepare data from the raw files produced by the microarray scanning apparatuses (either `.CEL` files for Affymetrix microarrays or `.txt` files for Agilent), you can use the `prepaffy` and `prepagil` commands, which preprocess Affymetrix and Agilent data, respectively.

The commands are called as follows. For `prepaffy`:

```
GATTACA prepaffy [-h | --help] [-r | --remove-controls]
    [--plot-size <x,y>] [--png] [--plot-number <int>]
    <input_dir> <output_file>
```

- The `-h | --help` flag ends the command early and prints a help message;
- The `-r | --remove-controls` flag, if set, causes the tool to filter out the control probes from the dataset. GATTACA does not give any other opportunity to do this.

- The `--plot-size <x,y>`, `--png` and `--plot-number <int>` flag can be used to customize the behaviour of plotting. `plot-size` controls the size of plots. If set, the `--png` flag causes the plots to be printed as `.png` format, and not the default of `.pdf`. The tool prints one plot per input file. The `plot-number` flag can limit the number of plots to `<int>`.
- `<input_dir>` must be replaced with the (full) path to the input folder.
- `<output_file>` must be replaced with the (full) path to the file that will be written with the expression data. The file will be formatted according to the `.csv` standard.

For `prepagil`:

```
GATTACA prepagil [-h | --help] [-r | --remove-controls]
    [-p | --grep-pattern <pattern>]
    [--plot-size <x,y>] [--png] [--plot-number <int>]
    <input\_dir> <output\_file>
```

For the `-h | --help`, `-r | --remove-controls`, `--plot-size <x,y>`, `--png` and `--plot-number <int>` flags, refer to the definitions above. The same applies for the `<input_dir>` and `<output_file>` arguments.

- The optional flag `-p | --grep-pattern <pattern>` overrides the default detection for the Agilent input files from `*.txt` to `<pattern>` (see the detailed descriptions below).

Both commands perform the following steps:

1. Find and load all input files;
   - `prepaffy` finds all files ending with the `.CEL` file extension in the target folder, and considers them as input. `prepagil`, instead, finds by default all `.txt` files. This can be customized in the call, however, as `.txt` is a common file extension.
2. Merge all inputs into a single expression file;
   - Both commands do this automatically, by collapsing probes referring to the same gene (the so-called "probe-set") to a single entry.
3. Make MA plots before normalization, as well as an overall expression boxplot (see section 6 for more information about the quality control plots);
4. Normalize the expression data;
   - The `prepaffy` command uses the Robust Multichip Average (RMA) procedure to normalize the data, while the `prepagil` uses a combination of background normalization and interarray correction provided by the `limma` package.
5. Make MA plots and an expression boxplot after the normalization, allowing to appreciate the effects of normalization;
6. If specified, remove control probes from the dataset;
7. `prepagil` collapses replicate probes by taking their mean value inside each sample;
8. The final datasets are saved to the output file.

The resulting file is ready to be analyzed by the other GATTACA commands, such as `run` or `annotate`.

4 Differential Gene Expression Analysis

The core functionality of GATTACA is to handle performing DEAs with a user friendly interface. Note that for very complex analyses it could be better to use GATTACA locally, by overriding some analysis steps. Refer to section 8 for more information.

The steps to take to perform a DEA with GATTACA are as follows:

1. Make a configuration file for the run;
2. Edit the configuration file to customize it to the required experimental design;
3. Run `gattaca run`.

In the following sections we explore how to execute these steps.

4.1 Create a config file

To create a configuration file, use the `gattaca init` command:

```
GATTACA init [-h | --help] <file_path>
```

Replace `<file_path>` with the (full) file path to where the config file should be created. The contents of the file may be changed at will.

4.2 Customize the config file

The configuration file created in the previous step follows the `yaml` format, whose specifications may be seen at the official `yaml` website[2]. A mini guide to `yaml` is available in section A.

The options are divided into sections. Each section controls different aspects of the analysis. The possible types of the various options are annotated in parentheses. The options available in GATTACA are:

- **`general`**: The general section contains general options to control plot size and type, whether to include annotations, and if to print extra data snippets during the analysis:
  - **`slowmode` (`bool`)**: If `true`, asks for confirmation after every analysis step. The default is `false`.
  - **`show_data_snippets` (`bool`)**: If `true`, prints extra snippets of data during the analysis. This is useful together with `slowmode` to see if the analysis is not doing anything wrong. The default is `false`.
  - **`save_pdf` (`bool`)**: If `true`, saves plots in `.pdf` format.
  - **`save_png` (`bool`)**: If `true`, saves plots in `.png` format.
  - **`plot_width` (`int`)**: The width of the plots, in inches.
  - **`plot_height` (`int`)**: The height of the plots, in inches.
  - **`png_resolution` (`int`)**: The pixels per inch of the png plots.
  - **`enumerate_plots` (`bool`)**: If `true`, each plot is marked with a number, in the order it is created.

[2] At the time of writing, the latest specification is `yaml 1.2.2`

  - **annotation_chip_id (str)**: The "chip id" string of the chip used to find the HUGO symbols of the respective probes. A table of the available chip ids can be seen in table 1.
- **switches**: The switches section contains various parameters to turn parts of the analysis on or off:
  - **dryrun (bool)**: Run the analysis, but do not save any output plots (with the exception of the log file). This can be useful to test out the analysis before committing to it, especially when used in combination with the `slowmode` and `show_data_snippets` options.
  - **renormalize (bool)**: Run quantile-quantile normalization on the data. This can be useful to normalize "unruly" samples, for which the normalization steps in `prepaffy` and `prepagil` are not enough. Additional plots are saved to appreciate this extra normalization step.
  - **limma (bool)**: Run DEA with `limma`.
  - **rankproduct (bool)**: Run DEA with `RankProduct`.
- **design**: The design section contains crucial options to set the experimental design that will be used to steer the analysis.
  - **experimental_design (str)**: The experimental design of the experiment. It must be a comma-delimited set of values, of the same length as the number of input samples, with each value being the label for the experimental variable of interest. Numbers specified at the end of each group can be used to represent sample pairings. For a detailed guide on how to define this parameter, see section 4.4.
  - **contrasts (list of str)**: A list of strings of the type `"group1-group2"`, where each "group" is a level in the experimental design. Each value in the list specifies a (different) contrast of interest. For a detailed guide on how to define this parameter, see section 4.4.
  - **batches (null OR str)**: If `null`, no batch effect will be corrected, assuming all samples derive from the same batch. If `str`, it is treated similar to the `experimental_design` string, and each level refers to a different batch. Note that `RankProd` cannot correct batch effects if, inside each batch, there are not at least than two samples for each experimental condition.
  - **extra_limma_vars (null OR list of str)**: If `null`, nothing happens. If `list of str`, each string in the list is treated similar to the `experimental_design` string, adding an additional variable to the `limma` analysis. This can be useful to control for additional confounding variables in the experiment.
  - **group_colors (list of str)**: A list of strings that can be understood by `R` to be a colour. Each colour will be paired with a different condition type in the experimental design, so at least that many colours must be specified. A list of possible colours can be found here.
  - **filters**: The parameters used by the analysis to filter the data:
    * **log2_expression (float)**: Filter out any genes that have lower $\log_2$ expression than this value. This value depends a lot on the experiment, and is generally higher for Agilent arrays. The default of "4" is good for Affymetrix arrays.
    * **fold_change (float)**: Filter out (mark as non-differentially-expressed) any

Chip ID	Microarray name
hgu133a	Affymetrix Human Genome U133 Set (A)
hgu133b	Affymetrix Human Genome U133 Set (B)
hgu133plus2	Affymetrix Human Genome HG-U133 Plus 2.0 Array
HsAgilentDesign026652	Agilent-026652 Whole Human Genome Microarray 4x44K v2
hugene10st	Affymetrix Human Gene 1.0-ST Array

Table 1: Chip IDs and the corresponding microarrays. The chip id codes must be used instead of the longer names when using GATTACA.

genes that have lower absolute Fold Change than this value. This is used to remove from the analysis all genes that, even if detected to be DEGs, as additional validation with other methods (such as PCR) would be impossible.

* **min_groupwise_presence (float)**: A value between 0 and 1, representing the proportion of samples in a single group of interest in which the fold_change filter threshold must be violated to be filtered out. If a gene passes the filter in at least one group, it is retained in the analysis. This allows for a more conservative filtering of the genes.

4.3 Running GATTACA

Once the options file is created and edited, the process of running gattaca is simple:

```
GATTACA run [-h | --help] <output_dir> <input_file> <options_file>
```

- Specifying the -h | --help command ends execution early, and prints usage information.
- <output_dir> must be replaced with the (full) path to the output directory of choice. Note that, since the docker instance is run as administrator, the final owner of the output folder will be the original one, regardless of whom started the command. Sub-folders are created in this path, so it is recommended to be empty.
- <input_file> must be replaced with the (full) path to the input file, a .csv file with the expression data, such as one created with prepaffy or prepagil. This file has one column per sample, and one row per gene. An additional column, named probe_id, contains the probe ids of the respective row names.
- <options_file> must be replaced with the (full) path to the options file used by the analysis.

4.4 Defining the experimental design

Defining the experimental design can seem daunting at first. Here are provided some guidelines on how to specify the relevant parameters in the analysis.

4.4.1 Design strings

A "design string" is a string that expresses the value of some categorical variable on each sample, with each variable separated by a comma (spaces are ignored). We will take

as example four samples named ”A”, ”B”, ”C” and ”D”, included in this order (from left to right) in an input file. Samples ”A” and ”C” are ”control” samples, while samples ”B” and ”D” are ”treated” samples. If we want to express this variable (the status of the samples) in a design string, we would write: `"control, treated, control, treated"`.

Design strings can be very repetitive, and very long if many samples are present in the analysis. Two shorthands are provided to alleviate this:

- `"( ... ):x"`: Values (`...`) in the parentheses are repeated sequentially ”`x`” number of times. For example: `"(a, b):2"` is equal to `"a, b, a, b"`.
- `"[ ... ]:x"`: Values (`...`) in the parentheses are repeated each a ”`x`” number of times. For example: `"[a, b]:2"` is equal to `"a, a, b, b"`.

Please note that these patterns **cannot** be nested, but they can be used together (`"(a, b):3, b, a, [a]:3, b"` is valid, for example). The same example from before can, therefore, be expressed with the equal string `"(control, treated):2"` instead.

4.4.2 The experimental design string

The experimental design string, specified in `design > experimental_design`, represent the groups in which the samples fall in. These are the groups of interest for the analysis, and will be the same groups specified in the `contrasts` parameter, determining between which samples DEGs have to be detected.

An additional variable that can be included in this same parameter is the sample pairings. Samples are paired if, for instance, they derive from the same patient, or the same tissue. This pairings can be expressed in the variable by adding a number, representing a certain pairing, at the end of each group name. For instance, consider six samples that fall into the ”control” and ”tumor” groups like such: `"control, tumor, control, tumor, control, tumor"`. The first two samples come from the same patient, as do the next two and so on. This pairing can be expressed in the following way: `"control1, tumor1, control2, tumor2, control3, tumor3"`. The actual numbers used are unimportant, as only the pairings of the samples are regarded.
The shorthand patterns support a way to specify these numbers automatically. By adding an asterisk (`*`) at the end of the value, it will be replaced with a progressively increasing number automatically. For example, the string `"control1, tumor1, control2, tumor2, control3, tumor3"` can be expressed as `"(control*, tumor*):3"`. `"[control*, tumor*]:2"` instead expands to `"control1, control2, tumor1, tumor2"`. Note that any numbers that are already present in the string will be skipped by the shorthand patterns. For instance, `"tumor4, (control*):2"` results is `"tumor4, control5, control6"`.

As numbers represent pairings, the group names cannot contain any numbers (even not in the end of the value). Additionally, either all samples must have a pairing annotation, or none may. If annotated, each different pairing must show at least one sample for each group variable.

4.4.3 Specifying technical replicates

Technical replicates are samples which are identical in all ways (experimental variables, batches, operator, etc...) and vary only due to different measurement runs. For instance, the same sample can be measured by several different chips to control technical variability.

Specifying them here will allow the analysis to take them into account. This allows for increased statistical power[3].

4.4.4 Specifying batches

Running the analysis in batches is often necessary, but introduces so-called *batch effects*, static differences in the expression measures due to technical variables (such as the operator that ran the analysis, the hybridization and exposure times, etc...). Batch effects have been handled in different ways in the literature. Nygaard *et al.* [3] offers a review of the subject, and GATTACA follows the recommended guidelines. When running the `limma` analysis, the data is corrected by TODO. When running `RankProd`, the batch effects are taken into account in the algorithm itself. Note that `RankProd` may only correct batch effects if for each grouping variable there are at least two samples in each batch.

The various batches can be specified with the `design > batches` variable.

4.4.5 Additional limma variables

`limma` can accept a virtually infinite amount of variables to control the tests. Extra variables may be submitted to `limma` in the `design > extra_limma_vars`, as a list of design strings, one string per additional variable.

4.5 Interpreting results

GATTACA outputs several files. Other than the quality control plots (whose meaning can be seen in section 6), `gattaca run` produces the following files:

- `correspondence_table.csv`: This file contains the correspondence between the original sample names (in the input files) and the new labels assigned by GATTACA, with the group names. It can be used to check the correspondence between the original samples and the experimental design vector.
- `DEG tables`: For each tool and each contrast, one DEGs table file is created, named with the pattern `<tool> - DEG Table <contrast>.csv`.
  - The `limma` DEGs tables have the following columns:
    * `probe_id`: The probe id related to the other results;
    * `logFC`: The computed $\log_2$ fold-change between the contrasted groups;
    * `AveExpr`: The average expression considering all samples;
    * `t`: The t statistic of the contrast;
    * `P.Value`: The original P-value originated from the `t` statistic;

[3] Note that not setting this variable when some samples are technical replicates is pseudo-replication, and can invalidate the analysis results.

        * adj.P.Val: The Benjamini-Hochberg adjusted p-value.
        * B: The B statistics represents how likely the gene is differentially expressed, in log-odds.
        * markings: The final marking of the gene, it being upregulated (1), downregulated (-1), or not differentially expressed (0). This is computed from the adjusted P-value with an $\alpha = 0.05$, and the logFC, taking in consideration the logfc filter.
        * SYMBOL: If a chip is specified, the SYMBOL column contains HUGO symbols of the respective probes.
    - The RankProd DEGs tables have the following columns:
        * probe_id: The probe id related to the other results;
        * gene.index: Arbitrary labelling of genes, set by RankProd before generating sample permutations;
        * logFC: The computed $\log_2$ fold-change between the contrasted groups;
        * RP/Rsum.UP: RankProd score for upregulated genes;
        * pfp.UP: Proportion of False Positives (PFP) for upregulated genes, treated similar to the limma adj.P.Val.
        * P.value.UP: P-value for upregulated genes, computed from the RankProd score.
        * RP/Rsum.DOWN: RankProd score for downregulated genes;
        * pfp.DOWN: PFP for downregulated genes, treated similar to the limma adj.P.Val.
        * P.value.DOWN: P-value for downregulated genes, computed from the RankProd score.
        * markings: The final marking of the gene, it being upregulated (1), downregulated (-1), or not differentially expressed (0). This is computed from the PFP values with an $\alpha = 0.05$, and the logFC, taking in consideration the logfc filter. The PFP values for upregulation are used for genes with positive logFC, and vice-versa.
        * AveExpr: The average expression considering all samples;
        * SYMBOL: If a chip is specified, the SYMBOL column contains HUGO symbols of the respective probes.

For most applications, the markings column can be used to retrieve up- and down-regulated genes in each contrast for each tool. Comparisons of the number of genes detected by the two tools are available in the quality control plots.

5 Annotating Results

Investigators are often interested in annotating their results with metadata. GATTACA provides the annotate command to annotate any .csv file with a probe_id column with annotations from Bioconductor databases for each chip.

The command is invoked as such:

```
GATTACA annotate [-h | --help] [-s | --select <selections>] <input_file>
                 <output_file> <chip_id>
```

- The -h | --help ends the command early and shows usage statistics.

- The `-s | --select <selections>` option can be used to select the metadata to annotate. The possible variables that can sourced from most databases are `ACCNUM`, `CHR`, `CHRLOC`, `CHRLOCEND`, `ENSEMBL`, `ENTREZID`, `ENZYME`, `GENENAME`, `GO`, `MAP`, `OMIM`, `PATH`, `PMID`, `REFSEQ`, `SYMBOL` and `UNIPROT`.
- `<input_file>` must be replaced with the (full) path to the input `.csv` file to annotate.
- `<output_file>` must be replaced with the (full) path to the output `.csv` file.
- `<chip_id>` must be replaced with a valid chip id. The supported chips are available in Table 1.

6 Quality control plots

This section provides explanations on how to interpret the quality control plots produced by the various commands, and additional details on how these are created.

6.1 MA plots

An MA plot is an application of the Bland-Altman plots for genomic data, where each gene is plotted as a point at M and A coordinates. The M coordinate for each gene is the log-ratio of the expression of that gene between two samples[4]. The A coordinate is the average expression of that gene in the two samples.
If one or both of the samples is instead a *group* of samples, the M coordinate is instead calculated on the log-ratio of the *groupwise median* of the expression of each gene is the sample groups. The "A" coordinate is the average expression between all the samples.
MA plots generated by GATTACA provide density shading, showing regions with gradually increasing point density from blue to red.

An example MA plot can be seen in Figure 1, from a spike-in experiment. It is generally expected to see a large number of genes around low "A" values, with low M spread. These are generally low-expressed genes, of which M values are non-zero due to biological variability. Genes that have high A values but low M values (on the center-right of the plot) are housekeeping genes. Genes that have high absolute M values are putative DEGs.

The overall trend of the plot is shown with a Generalized Additive Model (GAM) regression line, in red.

We expect that MA plots for normalized data to be roughly linear and centered (as in, the GAM line flat and centered on zero). Both `prepaffy`, `prepagil` make one MA plot for each sample, plotting it versus the median expression of all other samples. When `prepaffy` and `prepagil` produce MA plots, they are ordered and numbered from the most distorted to the least, allowing the user to detect at a glance possibly distorted samples.

If a sample remains distorted after normalization, a possible mode of action is to remove it from the analysis. Otherwise, one may specify the `renormalize` option in `gattaca run` to perform quantile-quantile normalization on the data, forcing distorted samples to normality. Note however that doing this might also distort the results of the analysis.

[4]When reading "Group 1 vs Group 2" it is assumed that the M value is calculated as $\log_2(Group2) - \log_2(Group1)$

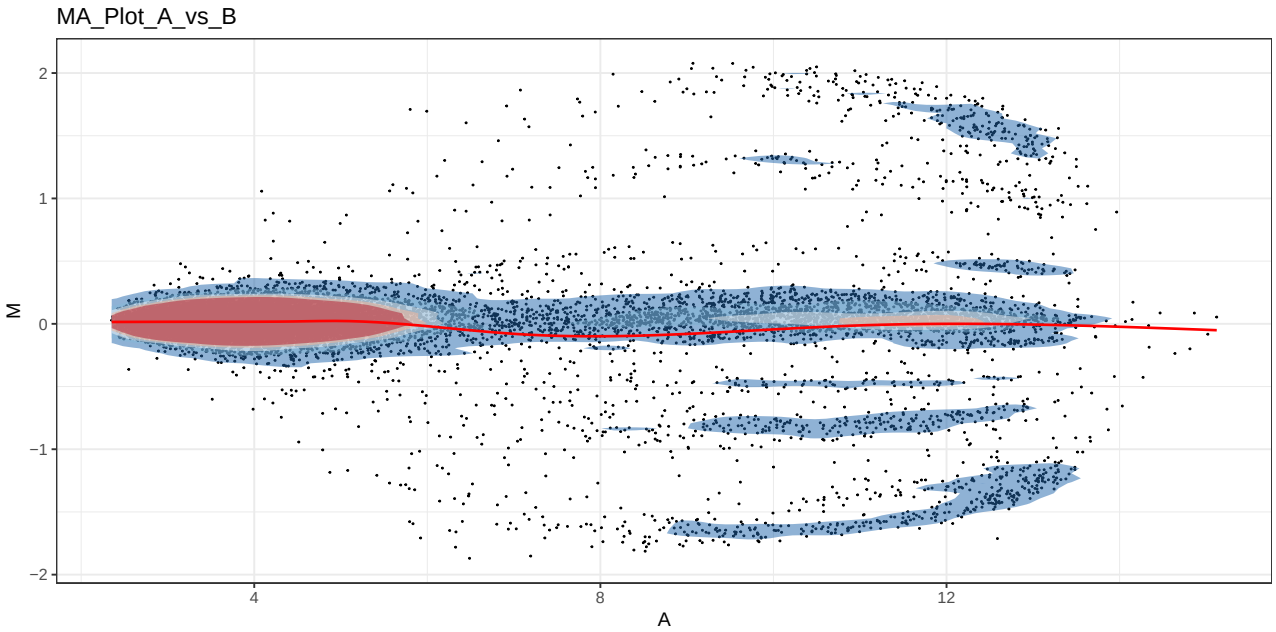

Figure 1: MA plot constructed with spike-in data from Zhu *et al.* [4]. This MA plot shows the group of samples labelled as the "A" condition vs the group of samples labelled as the "B" condition.

The `run` command also produces MA plots, with the purpose of highlighting differences between the groups of interest during DEA, as well as showing detected DEGs in a plot. An example of such an annotated plot is shown in Figure 2

6.2 Expression Boxplots

Each boxplot in an expression boxplot plot represents a different sample, with each point representing the expression value of each gene. An example of such a plot is shown in Figure 3. We expect that the general shape of all boxplots to be roughly similar, especially after normalization. If one or more samples have different shapes than the others, it is possible to employ the same procedures proposed in section 6.1: use the `renormalize` option in `gattaca run`, or eliminate the non-homogeneous sample(s).

6.3 Dendrogram, PCA and Scree plots

Hierarchical clustering dendrograms (hereby referred to as "dendrograms"), Principal Component Analysis (PCA) plots and Scree plots allow the visual detection of clustering of the data.

Dendrograms show the hierarchical distances between the samples. Samples connected more closely than others are more clustered. An example of such a plot can be seen in Figure 4.

PCA plots show the result of the PCA analysis of the data. The main PCA plot shows principal component 1 vs principal component 2 of the samples. An example of such a plot

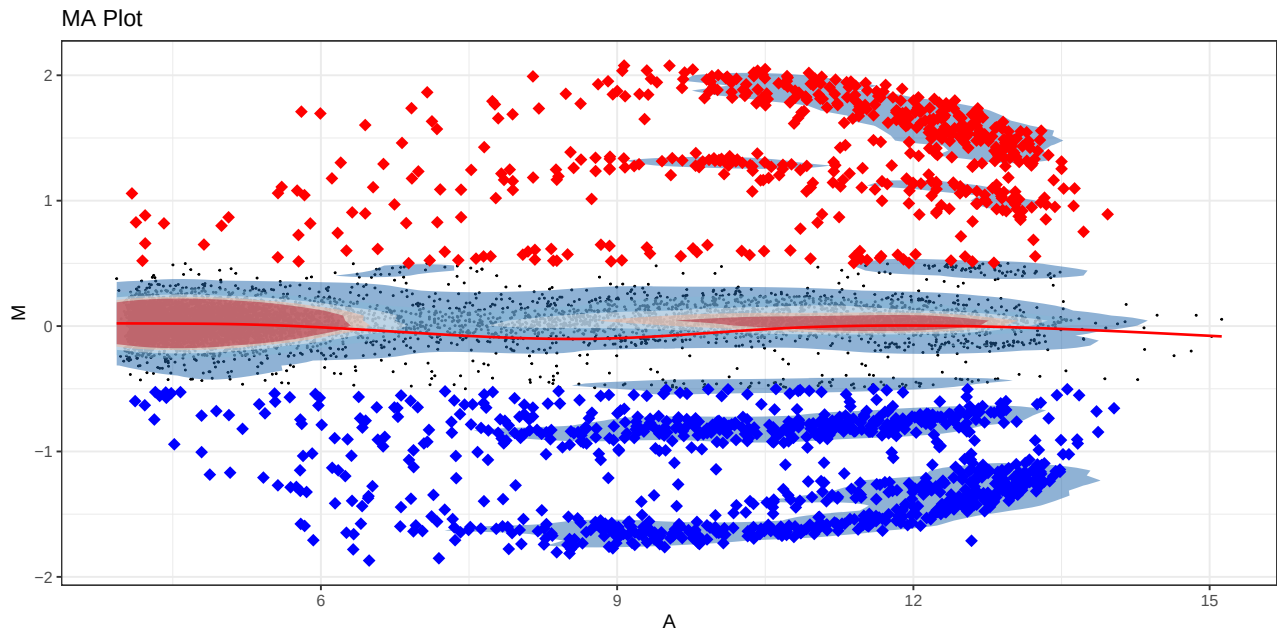

Figure 2: MA plots with added DEGs detected by the `limma` package by comparing conditions "B-A". This plot was generated with the `gattaca run` command on the spike-in Data from Zhu *et al.* [4]. Genes highlighted in Red are detected to be significantly overexpressed in B in respect of A, while genes marked in Blue are similarly underexpressed in B in respect of A. That is, red-labelled genes are overexpressed DEGs, while blue-labelled genes are underexpressed DEGs.

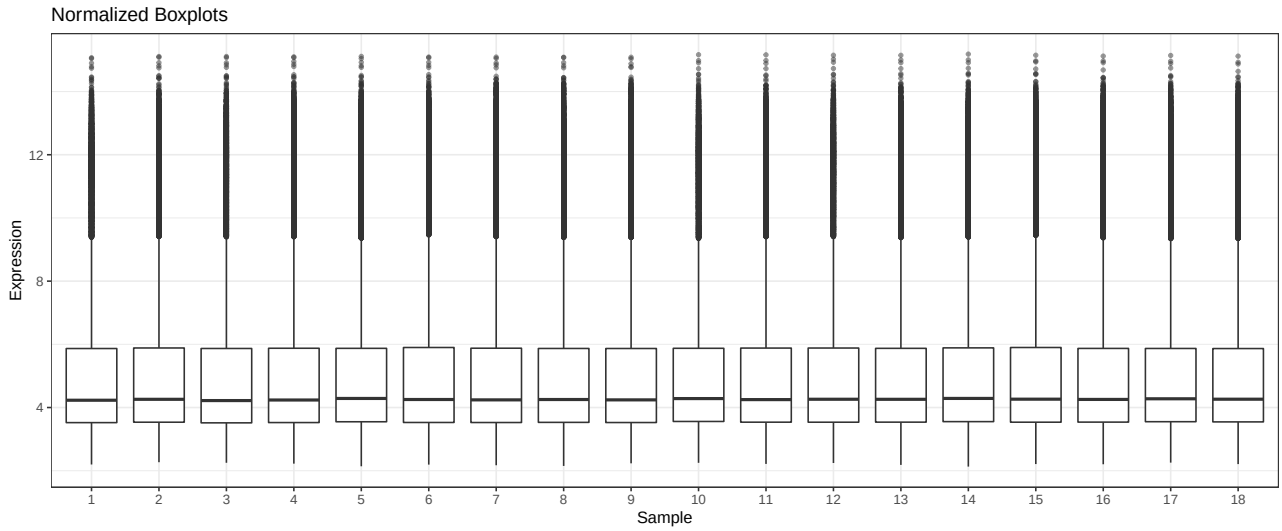

Figure 3: Normalized expression boxplot. This plot was generated with the `prepaffy` command on the spike-in Data from Zhu *et al.* [4]. Each boxplot refers to a sample. The boxplots are numbered, and are in the same order as the input files.

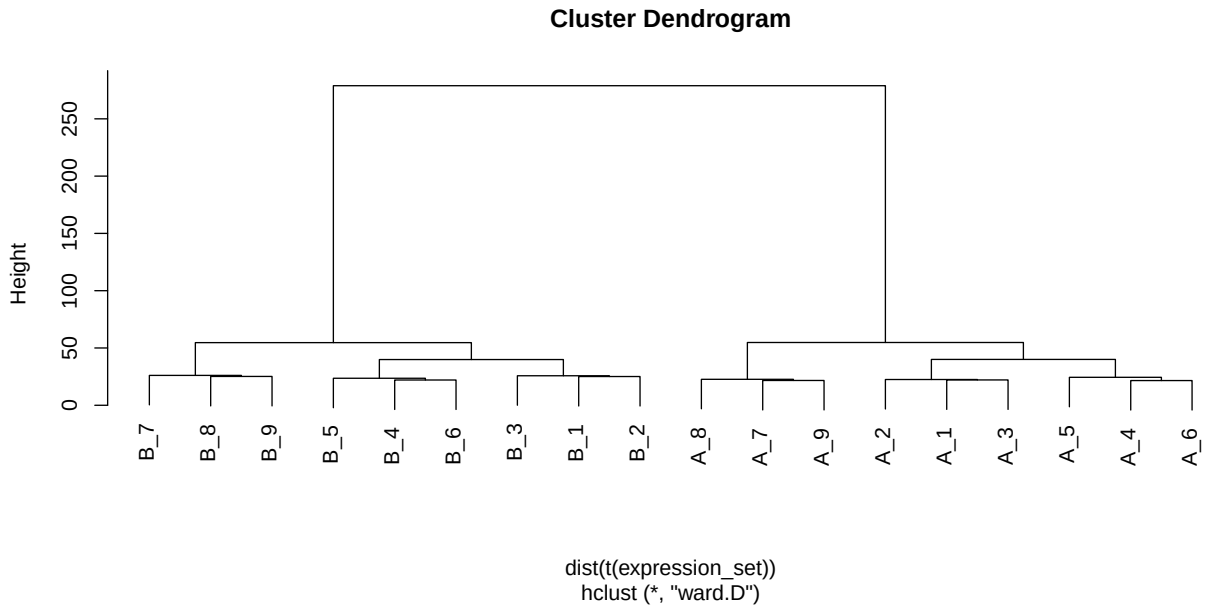

Figure 4: A hierarchical clustering dendrogram plot of spike-in data from Zhu *et al.* [4]. The data shows very large distances between the ”A” and ”B” samples, with smaller distances inside each group.

can be seen in Figure 5. An additional collection of PCA plots, named ”PCA pairs”, can also be found, and show plots generated from the combinations of other principal components.

Closely associated with the PCA plots is the Scree plot. The Scree plot shows the variability captured by each principal component, as well as the rolling sum of the captured variability. An example Scree plot can be seen in Figure 6. It is useful to consider the Scree plot together with the PCA plots. If principal components cannot capture most of the variability of the data, the PCA plots are less informative.

We expect the samples to be homogeneously distant from each other, with any obvious clustering a sign of a possible batch effect. The main exception from this rule of thumb is the clustering of different types of samples, for instance between the conditions of interest. While such a clustering (especially if very dramatic) could be due to batch effects that are not of interest, they could be a sign of important effects of the actual conditions. Therefore, light clustering of samples with the same conditions of interest can be normal.

6.4 Poisson plots

Poisson plots, or SD vs Mean plots, show the relationship between the standard deviation and the average of all genes in a certain group or groups. An example of such a plot can be seen in Figure 7.

The data is expected to be roughly Poissonian, with the standard deviation positively correlated with the mean. Any distortion or artifacts in these plots could be symptoms of deeper problems in the array hybridization, exposure, or both. The specific problem should be identifiable from the other quality-control plots.

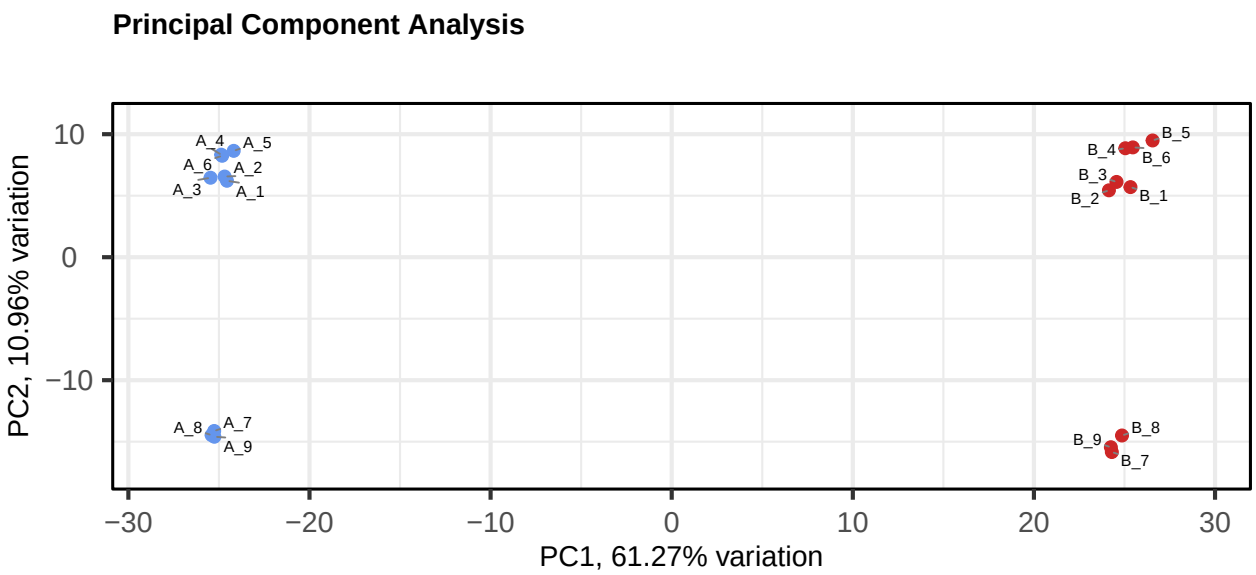

Figure 5: PCA plot of spike-in data from Zhu *et al.* [4]. The data shows similar clustering as in Figure 4, however, additional batch effects become clearer inside the two sample types.

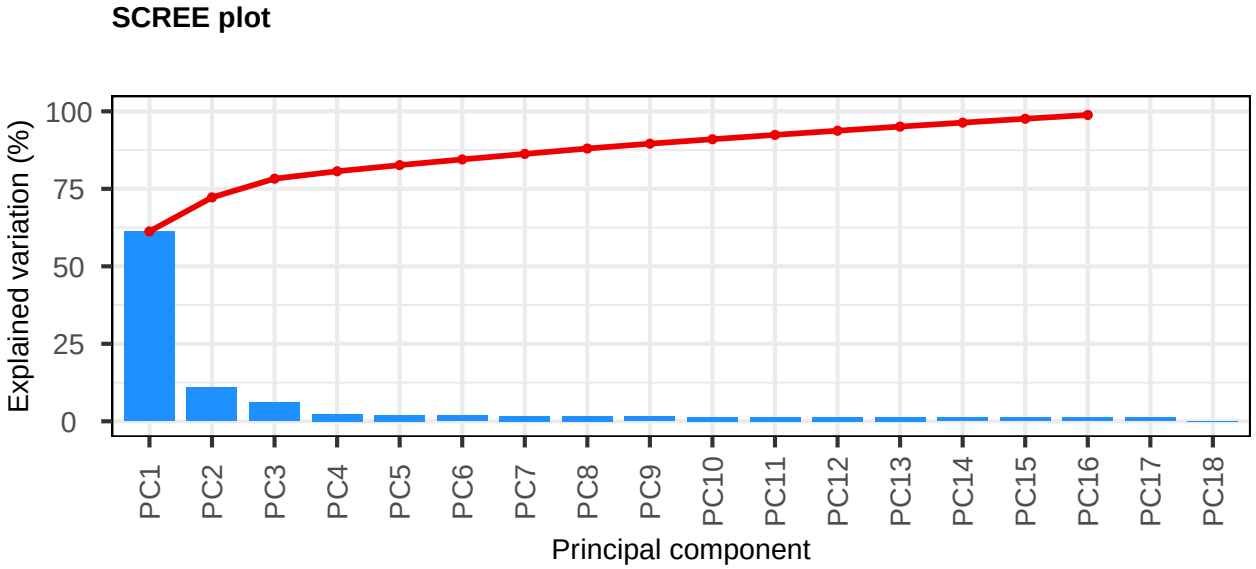

Figure 6: Scree plot of spike-in data from Zhu *et al.* [4]. As the first principal component captures most of the variability of the data, the PCA plots are very informative.

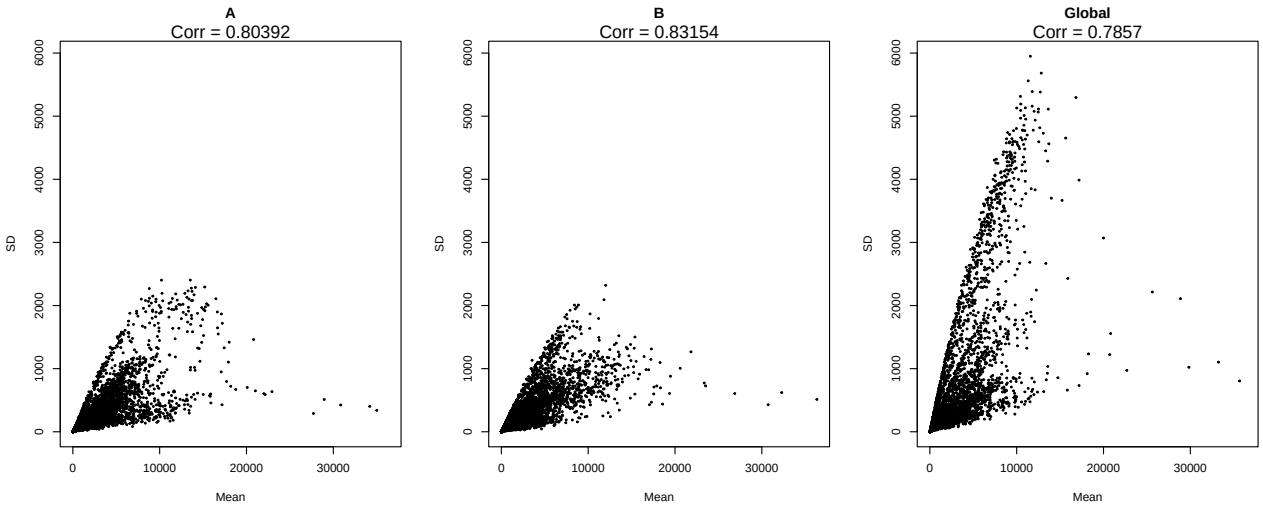

Figure 7: A Poisson plot generated from the spike-in data from Zhu *et al.* [4]. As expected, the data is roughly Poissonian, that is, the standard deviation is positively correlated with the mean. The same plot is drawn for the samples of condition "A", condition "B", and the summary of all samples ("Global").

6.5 Volcano Plots

Volcano plots are useful to show the result of a DEA. The Y axis shows the p-values associated to each gene in a certain contrast (such as group "B" vs group "A"), but in $\log_{10}$ scale, to allow a wide range of p-values to be plotted. The X axis shows the respective $\log_2$ fold-changes of each gene.

Threshold lines are drawn on the plot: an horizontal line representing the p-value threshold for statistical significance, and two vertical lines showing the $\log_2$ fold change thresholds to be considered DEGs. Therefore, all genes that fall on the upper-left and upper-right quadrants are downregulated and upregulated DEGs, respectively.

Do note that the Y axis shows P-values, which are not corrected for multiple hypothesis testing. It is the P-value threshold that is moderated, instead, so the highlighted genes are the same ones that would be shown by plotting False Discovery Rate (FDR) or PFP values on the Y axis and using a threshold of 0.05. The only difference is graphical: P-values show the genes as more spread-out, while corrected values are squashed.

Volcano plots created by GATTACA can be labelled with HUGO symbols, if an annotation database is specified when calling `gattaca run`. If unspecified, the probe ids are used instead (as seen in Figure 8).

7 Computational reproducibility

GATTACA allows the investigators to easily achieve computational reproducibility. Only a few items have to be recorded (and shared) to allow an identical run even far in the future. These are:

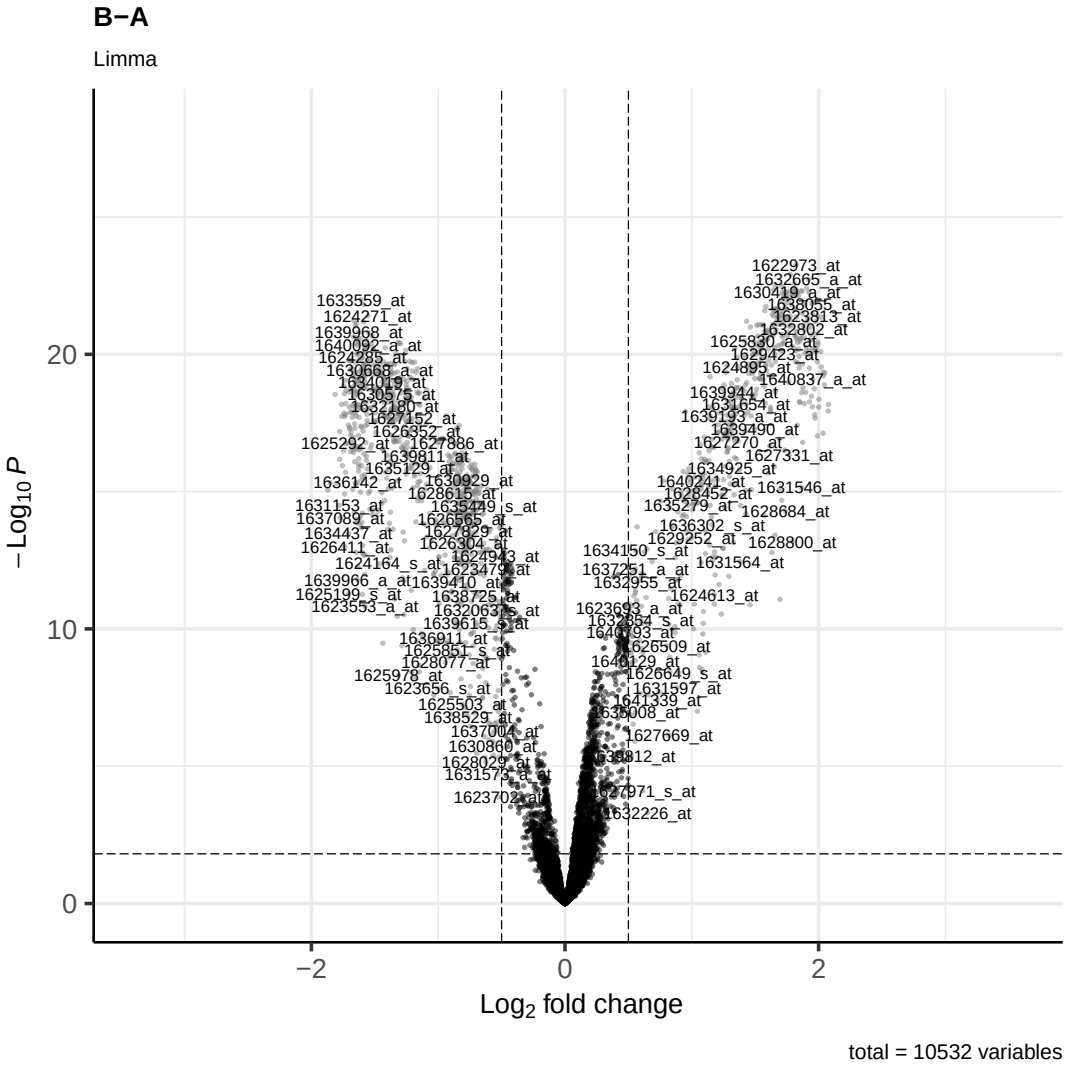

Figure 8: Volcano plot of the results of the `limma` analysis on the spike-in dataset from Zhu *et al.* [4]. The lower horizontal dotted line represents the corrected p-value equal to an FDR threshold of 0.05. The vertical lines represent the $\log_2$ fold change threshold of the analysis. The genes in the upper-left and upper-right quadrants are downregulated and upregulated DEGs, respectively.

- Any input files;
- The version of the docker used;
- For `prepaffy` and `prepagil`, it is sufficient to save the call to the tool.
- For `run`, recording the options file used in the analysis is sufficient to reproduce it identically. In this case, the call to the tool has no influence to the run.
- For `annotate`, it is sufficient to save the call to the tool.

8 Install and use GATTACA interactively

GATTACA is not distributed as a standalone package on CRAN. However, it could be useful in more complex analyses to access its functions directly in a non-dockerized environment. To do this, clone the repository locally (you must have `git` installed):

```
git clone https://github.com/Feat-FeAR/GATTACA.git
```

This will clone the repository locally into a new `GATTACA/` folder. Copy the `GATTACA/src` folder in your own project. Install the dependencies needed by GATTACA in your local R installation or virtual environment by following the CONTRIBUTING.md guide on GitHub. Finally, start R, change your working directory to be inside of the folder that contains `src/` (such as inside `../src`), and source the `src/__init__.R` file before starting the analysis. Afterwards, you may source any other module at will, calling GATTACA functions interactively. Refer to the various help messages and comments in the source code for additional help.

8.1 Contributing

The same steps can be used to contribute to GATTACA. For more information, refer to the CONTRIBUTING.md guide on GitHub. Every contribution is welcome.

The GitHub issues page has information on the current issues and improvements that require contributions.

Bibliograpy

1. Ritchie, M. E. *et al.* Limma Powers Differential Expression Analyses for RNA-Sequencing and Microarray Studies. *Nucleic Acids Research* **43,** e47. ISSN: 0305-1048 (Apr. 2015).
2. Hong, F. *et al.* RankProd: A Bioconductor Package for Detecting Differentially Expressed Genes in Meta-Analysis. *Bioinformatics* **22,** 2825–2827. ISSN: 1367-4803 (Nov. 2006).
3. Nygaard, V., Rødland, E. A. & Hovig, E. Methods That Remove Batch Effects While Retaining Group Differences May Lead to Exaggerated Confidence in Downstream Analyses. *Biostatistics* **17,** 29–39. ISSN: 1465-4644 (Jan. 2016).
4. Zhu, Q., Miecznikowski, J. C. & Halfon, M. S. Preferred Analysis Methods for Affymetrix GeneChips. II. An Expanded, Balanced, Wholly-Defined Spike-in Dataset. *BMC Bioinformatics* **11,** 285. ISSN: 1471-2105 (May 2010).

Appendices

A Mini YAML guide

The possible `yaml` types are listed here. Note that this section covers only the types used in the GATTACA options, not the whole `yaml` specification.

- Boolean type (`bool`): `true` or `false`.
- Strings (`str`): Any character string surrounded by double or single quotes. Example: `"A string"`, `"12.4 is my favourite number."`.
- Integers (`int`): Any number without a fractional part, like `1` or `42`.
- Floats (`float`): Any number with a fractional part, like `1.0` or `3.14159265359`. Any number of digits after the comma are allowed.
- Lists (`list`): A list contains a series of other values, of any type (including lists). Lists of values with just one type are annotated with `list of <type>`. It is created by square brackets, with the single values separated by commas. Example: `[1, 2, 3]` would be a `list of int`, `["banana", "papaya"]` would be a `list of str`.
- Nothing (`` `null` ``): Many options allow to be unset. This is done by specifying `null`, meaning ”no value”.