

End-to-end analysis of cell-based screens: from raw intensity readings to the annotated hit list

Michael Boutros, Lígia Brás, Florian Hahne and Wolfgang Huber

October 29, 2019

Contents

1	Introduction	2
2	Reading the intensity data	3
2.1	Importing intensity data files with other formats	5
3	The <i>cellHTS</i> class and reports	6
4	Screen configuration: annotating the plate results	10
4.1	Format of the plate configuration file	12
4.1.1	Multiple plate configurations	13
4.2	Format of the screen log file	15
5	Normalization, scoring and summarization of replicates	15
6	Probe annotation	20
6.1	Adding additional annotation from public databases	21
6.1.1	Installation	22
6.1.2	Using biomaRt to annotate the target genes online	22
7	Report	25
7.1	Controlling settings	29
7.2	Exporting data to a tab-delimited file	29
8	Category analysis	30
9	Comparison with the results previously reported	36

10 Appendix: How to convert <i>cellHTS</i> to <i>cellHTS2</i> configuration files	36
11 Appendix: Normalization methods implemented in <i>cellHTS2</i> package	41
11.1 Controls-based normalization	41
11.1.1 Percent of control	41
11.1.2 Normalized percent inhibition	42
11.2 Non-controls-based normalization	42
11.2.1 Z score method	42
11.2.2 Plate median normalization	43
11.2.3 B score method	43
12 Appendix: Data transformation	44
13 Session info	46

1 Introduction

The package *cellHTS2* is a revised and improved version of the *cellHTS* package ¹.

This report describes the structure of the *cellHTS* class, while explaining all the steps necessary to run a complete analysis of a cell-based high-throughput screen (HTS), from raw intensity readings to an annotated hit list.

This text has been produced as a reproducible document [6]. It contains the actual computer instructions for the methods it describes, and these in turn produce all results, including the figures and tables that are shown here. The computer instructions are given in the language R, thus, in order to reproduce the computations shown here, you will need an installation of R (version 2.3 or greater) together with a recent version of the package *cellHTS2* and of some other add-on packages. Within R, the following commands can be used to install *cellHTS2* along with all dependent packages.

To reproduce the computations shown here, you do not need to type them or copy-paste them from the PDF file; rather, you can take the file *cellhts2Complete.Rnw* in the *scripts* directory of the package, open it in a text editor, run it using the R command *Sweave*, and modify it to your needs.

¹To convert a S3 class *cellHTS* object that was made using the *cellHTS* package into an S4 class *cellHTS* object suitable for *cellHTS2*, please see the function *convertOldCellHTS*.

First, we load the package.

```
> library("cellHTS2")
```

2 Reading the intensity data

We consider a cell-based screen that was conducted in microtiter plate format, where a library of double-stranded RNAs was used to target the corresponding genes in cultured *Drosophila* *Kc*₁₆₇ cells [2]. Each of the wells in the plates contains either a gene-specific probe, a control, or it can be empty. The experiments were done in duplicate, and the viability of the cells after treatment was recorded by a plate reader measuring luciferase activity. The promoter has been chosen such that the luciferase activity is indicative of ATP levels. Although this set of example data corresponds to a single-channel screening assay, the *cellHTS2* package can also deal with cases where there are readings from more channels, corresponding to different reporters.

Usually, the measurements from each replicate and each channel come in individual result files. The set of available result files and the information about them (which plate, which replicate, which channel) is contained in a spreadsheet, which we call the *plate list file*. This file should contain at least the following columns: *Filename*, *Plate* and *Replicate*. The last two columns should be integer numbers, with values ranging from 1 to the maximum number of plates or replicates, respectively. An optional *Batch* column can be used to provide batch information about the experiment, i.e., changes in reagents, or days for a multi-day experiment. See Section [refsec:multPlateConfs](#) for more details. The first few lines of an example plate list file are shown in [Table 1](#), while [Table 2](#) shows the first few lines from one of the plate result files listed in the plate list file.

The first step of the analysis is to read the plate list file, to read all the intensity files, and to assemble the data into a single R object that is suitable for subsequent analyses. The main component of that object are arrays with the intensity readings of all plates, channels, and replicates. We demonstrate the R instructions for this step. First we define the path where the input files can be found.

```
> experimentName <- "KcViab"  
> dataPath <- system.file(experimentName, package="cellHTS2")
```

In this example, the input files are in the *KcViab* directory of the *cellHTS2* package. To read your own data, modify *dataPath* to point to the directory

Filename	Plate	Replicate
FT01-G01.txt	1	1
FT01-G02.txt	1	2
FT02-G01.txt	2	1
FT02-G02.txt	2	2
FT03-G01.txt	3	1
...

Table 1: Selected lines from the example plate list file `Platelist.txt`.

FT01-G01	A01	887763
FT01-G01	A02	958308
FT01-G01	A03	1012685
FT01-G01	A04	872603
FT01-G01	A05	1179875
...

Table 2: Selected lines from the example signal intensity file `FT01-G01.txt`.

where they reside. We show the names of 12 files from our example directory:

```
> dataPath
[1] "/tmp/RtmpZNPvIz/Rinst66fe18ca5750/cellHTS2/KcViab"
> rev(dir(dataPath))[1:12]
[1] "old-Screenlog.txt"      "old-Plateconf.txt"
[3] "Screenlog.txt"         "Platelist.txt"
[5] "Plateconf.txt"         "GeneIDs_Dm_HFA_1.1.txt"
[7] "FT57-G02.txt"          "FT57-G01.txt"
[9] "FT56-G02.txt"          "FT56-G01.txt"
[11] "FT55-G02.txt"          "FT55-G01.txt"
```

and read the data into the object `x`

```
> x <- readPlateList("Platelist.txt",
+                   name=experimentName,
+                   path=dataPath)
> x
```

```

cellHTS (storageMode: lockedEnvironment)
assayData: 21888 features, 2 samples
  element names: Channel 1
phenoData
  sampleNames: 1 2
  varLabels: replicate assay
  varMetadata: labelDescription channel
featureData
  featureNames: 1 2 ... 21888 (21888 total)
  fvarLabels: plate well controlStatus
  fvarMetadata: labelDescription
experimentData: use 'experimentData(object)'
state:  configured = FALSE
       normalized = FALSE
       scored = FALSE
       annotated = FALSE
Number of plates: 57
Plate dimension: nrow = 16, ncol = 24
Number of batches: 1

```

The plate format used in the screen (96-well or 384-well plate design) is automatically determined from the raw intensity files when calling the *readPlateList* function.

2.1 Importing intensity data files with other formats

The function *readPlateList* has the argument `importFun` that can be used to provide a different import function to read plate result files with a format different from that shown in Table 2. For example, to import plate data files from an EnVision plate reader, set `importFun=getEnVisionRawData` or `importFun=getEnvisionCrosstalkCorrectedData` when calling *readPlateList*. Please see the help page of function *getEnVisionRawData* for an example. Another import function (“importData.R”) is given together with the example data set for a enhancer-suppressor screen in the directory called *TwoWayAssay* of this package.

While for the above data sets, the measurements from each replicate and channel come in separate result files, this is not the case when measurement files are provided in the HTanalyst format. In this case, each output file contains meta-experimental data together with intensity readings (in a

matrix-like layout) of a set of plates made for the same replicate or screen. Thus, there is no need to have a plate list file, and instead of using *readPlateList* we should call the function *readHTAnalystData*. Please see the help page for this function (`? readHTAnalystData`), where we illustrate how it can be applied to import HTAnalyst data files.

3 The *cellHTS* class and reports

The basic data structure of the package is the class *cellHTS*, which is a container for cell-based high-throughput RNA interference assays (data and experimental meta-data) performed in multi-plate format. This class extends the class *NChannelSet* of the *Biobase* package [7].

The data can be thought of as being organised in a two- or three-dimensional array as follows:

1. The first dimension corresponds to reagents (e.g. siRNAs, chemical compounds) that were used in the assays. For example, if the screen used 100 plates of 384 wells (24 columns, 16 rows), then the first dimension has size 38,400, and the *cellHTS* object keeps track of plate ID, row, and column associated with each element. For historic reasons, and because we are using infrastructure that was developed for microarray experiments, the following terms are used synonymously for the elements of the first dimension: *reagents*, *features*, *probes*, *genes*.
2. The second dimension corresponds to assays, including replicates and different experimental conditions (cell type, treatment, genetic background). A potentially confusing terminology is that the data structure that annotates the second dimension is called **phenoData** (see below). This is because we are using infrastructure, namely the *NChannelSet* class from the *Biobase* package, that uses this unfortunate term for this purpose. Sometimes, the elements of this second dimension are also called *samples*.
3. The (optional) third dimension corresponds to different channels (e.g. different luminescence reporters)

The main structures contained in a *cellHTS* object are:

assayData an object of class *AssayData*, usually an environment containing a set of matrices of identical size. Each matrix represents a single channel. In each matrix, the rows correspond to features or reporters

(e.g. siRNAs, dsRNAs) and the columns to samples (different conditions and/or replicates).

phenoData a dataframe (more precisely, an object of class *Annotated-DataFrame*) containing information about the screens, such as the replicate number and the type of biological assay. It must have the following columns in its data component:

replicate a vector of integers giving the replicate number

assay a character vector giving the name of the biological assay or condition

Both of these columns have the same length as the number of samples in the **assayData** slot. The choice of name *phenoData* for this structure is unfortunate, it has nothing to do with phenotypes.

featureData a dataframe (more precisely, an object of class *Annotated-DataFrame*) that contains information about the reagents used in the experiment. There are three mandatory columns, in addition there can be an arbitrary number of additional columns, for example a target gene identifier. The mandatory columns are:

plate integers specifying the plate number (e.g. 1, 2, ...)

well alphanumeric character strings giving the well ID within the plate (e.g. A01, B01, ..., P24).

controlStatus a factor specifying the annotation for each well with possible levels: *empty*, *other*, *neg*, *sample*, *pos*. Other levels besides *pos* and *neg* may be employed for the positive and negative controls.

plateData a list of dataframes, where each list item contains plate-specific information. The structure of the individual dataframes is fixed: rows are supposed to be plates and columns are supposed to be samples (i.e., the number of columns has to be the same as the number of columns in the **assayData** matrices. By default, the only available list item is **Batch** containing the information about experimental batches in the experiment. Unless explicitly specified in the plate list file, these will all be set to 1.

experimentData an object of class *MIAME* containing descriptions of the experiment.

The *cellHTS* class also includes additional slots that are used to store the input files used to assemble the *cellHTS* object. These are:

plateList a dataframe containing names and metadata about input measurement data files, plus a column named *status*, of type *character*, whose elements are the string "OK" if the data import appeared to have gone well, and the respective error or warning message otherwise.

intensityFiles a list whose components are copies of the imported input data files. Its length corresponds to the number of rows of **plateList**.

plateConf a data frame containing what was read from the configuration file for the experiment (except the first two header rows).

screenLog a data frame containing what was read from the screen log file for the experiment (in case there was one).

screenDesc a character containing what was read from the description file of the experiment.

Other slots are `rowcol.effects`, `overall.effects` and `annotation`. For a detailed description of this class, please type `class ? cellHTS`.

In Section 2, we created the object `x`, which is an instance of the *cellHTS* class. The measurements intensities are stored in the slot `assayData` of `x`. The slot called `state` helps to keep track of the preprocessing state of our *cellHTS* object. This slot can be accessed through the *state*, as shown below:

```
> state(x)

configured normalized      scored  annotated
      FALSE      FALSE      FALSE      FALSE
```

It contains a logical vector of length 4 representing the processing status of the object. It should have the names "configured", "normalized", "scored" and "annotated". We can thus see that `x` has not been configured, annotated, normalized or scored yet.

These 4 main stages are explained along this vignette and can be briefly described as follows:

Configuration involves annotating the experiment with information about the screen (e. g. title, when and how it was performed, which organism,

which library, type of assay, etc.), annotating the measured values with information on controls and flagging invalid measurements. This step is covered in Section 4 and is prerequisite for preprocessing the experimental screening data (i. e. for normalization and scoring).

Annotation involves annotating the features (reagents) with information about, for example, their target genes. This step, detailed in Section 6, is not essential for preprocessing, but this information can be used for the HTML quality reports generated by *cellHTS2* and in further analyses (see the complete report, as explained in Section 1).

Normalization involves removing systematic variations, making measurements comparable across plates and within plates, enhancing the biological signal and eventually transforming the data to a scale suitable for subsequent analyses.

Replicates scoring and summarization involves standardizing the values for each replicate and then summarizing replicate values in order to obtain a single-value per probe (for example, a robust *z*-score).

The steps of data normalization, replicate scoring and summarization constitute the preprocessing work-flow of the screening data. They alter the contents of `assayData` slot and even the number of channels of the *cellHTS* object. The complete analysis project is contained in a set of *cellHTS* objects that reflect different preprocessing stages and that can be shared with others and stored for subsequent computational analyses.

The *cellHTS2* package offers export functions for generating human-readable reports, which consist of linked HTML pages with tables and plots. The final scored hit list is written as a tab-delimited format suitable for reading by spreadsheet programs.

Returning to our example data set, we create a report using the function *writeReport*. Since until now we only have unnormalized experimental data, the *cellHTS* object should be given to the function as the `raw` argument:

```
> out <- writeReport(raw=x)
```

This will write the report into the directory given by the argument `outdir`. The option `force=TRUE` tells the function to delete and overwrite a possibly already existing directory of the same name. This option needs to be used with caution and is only accepted if `outdir` is explicitly specified. If the argument `outdir` is not specified, the default is a subdirectory of the current working directory with name given by `name(x)`

Wells:	384	
Plates:	57	
Plate	Well	Content
*	*	sample
*	A0[1-2]	other
*	B01	neg
*	B02	pos

Table 3: Selected lines from the example plate configuration file `Plate-conf.txt`.

In order to keep a reference to the R commands used to create the HTML output, one can provide the path to an ASCII script file using the `mainScriptFile` argument. This comes back to the notion of reproducible research mentioned before, and we strongly suggest to make use of this feature. In fact, the function will issue a warning if no script file is supplied. This said, we will not make use of the `mainScriptFile` arguments, since we don't really have scripting files at hand; all commands are contained in this document.

It can take a while to run this function, since it creates a large number of graphics files. After this function has finished, the index page of the report will be in the file indicated by the variable `out`,

```
> out
[1] "/tmp/RtmpZaG5rB/index.html"
```

and you can view it by directing a web browser to that file.

```
> if (interactive()) browseURL(out)
```

4 Screen configuration: annotating the plate results

The next step of the analysis involves reading and processing three input files specific of the screening experiment:

- *Screen description file* contains a general description of the screen, its goal, the conditions under which it was performed, references, and any other information that is pertinent to the biological interpretation

Plate	Sample	Well	Flag	Comment
6	1	A01	NA	Contamination
6	2	A01	NA	Contamination
6	1	A02	NA	Contamination
...

Table 4: Selected lines from the example screen log file `Screenlog.txt`.

of the experiments. In *cellHTS2* package we provide a function that creates a template description file whose entries can be edited and completed by the user. Type `? templateDescriptionFile`. This file contains the entries compliant with the *MIAME* class and also additional fields specific for the *cellHTS* class.

- *Plate configuration file* is used to annotate the measured data with information on controls. The content of this file for the example data set analysed here is shown in Table 3 and the expected format for this file is explained in Section 4.1.
- *Screen log file* (optional) is used to flag individual measurements as invalid. The first 5 lines of this file are shown in Table 4, and the layout for the *screen log file* is detailed in Section 4.2.

To apply the information contained in these three files in our *cellHTS* object, we call:

```
> x <- configure(x,
+               descripFile="Description.txt",
+               confFile="Plateconf.txt",
+               logFile="Screenlog.txt",
+               path=dataPath)
```

Note that the function *configure*² takes `x`, the result from Section 2, as an argument, and we then overwrite `x` with the result of this function. If no screen log file is available for the experiment, the argument `logFile` of the function *configure* should be omitted.

²More precisely, *configure* is a method for the S4 class *cellHTS*.

4.1 Format of the plate configuration file

The software expects this to be a rectangular table in a tabulator delimited text file, with mandatory columns *Plate*, *Well*, *Content*, plus two additional header lines that give the total number of wells and plates (see Table 3 for an example). The content of this file (except the two header lines) are stored in slot `plateConf` of `x`.

As the name suggests, the *Content* column provides the content of each well in the plate (here referred to as the *well annotation*). Mainly, this annotation falls into four categories: empty wells, wells targeting genes of interest, control wells, and wells containing other things that do not fit in the previous categories. The first two types of wells should be indicated in the *Content* column of the plate configuration file by *empty* and *sample*, respectively, while the last type of wells should be indicated by *other*. The designation for the control wells in the *Content* column is more flexible. By default, the software expects them to be indicated by *pos* (for positive controls), or *neg* (for negative controls). However, other names are allowed, given that they are specified by the user whenever necessary (for example, when calling the *writeReport* function). This versatility for the control wells' annotation is justified by the fact that, sometimes, multiple positive and/or negative controls can be employed in a given screen, making it useful to give different names to the distinct controls in the *Content* column. Moreover, this versatility might be required in multi-channel screens for which we frequently have reporter-specific controls.

The *Well* column contains the name of each well of the plate in alphanumeric format (in this case, A01 to P24), while column *Plate* gives the plate number (1, 2, ...). These two columns are also allowed to contain regular expressions. In the plate configuration file, each well and plate should be covered by a rule, and in case of multiple definitions only the last one is considered. For example, in the file shown in Table 3, the rule specified by the first line after the column header indicates that all of the wells in each of the 57 assay plate contain "sample". However, a following rule indicate that the content of wells A01, A02 and B01 and B02 differ from "sample", containing other material (in this case, "other" and controls).

Note that the well annotations mentioned above are used by the software in the normalization, quality control, and gene selection calculations. Data from wells that are annotated as *empty* are ignored, i. e. they are set to `NA`.

Here we look at the frequency of each well annotation in the example data:

```
> table(wellAnno(x))

sample other  neg  pos
21660   114   57  57
```

We can also use the function `configurationAsScreenPlot` to see the plate configuration of all of the plates as a screen plot:

```
> configurationAsScreenPlot(x)
```

The result is shown in Figure 1. This can be useful to verify the correctness of the plate configuration when a complex set of rules with regular expressions are used.

A special case of well annotation is when different types of positive controls are used for the screening, that is *enhancer* and *suppressor* compounds. The vignette *Analysis of screens with enhancer and suppressor controls* accompanying this package explains how such controls can be handled during the screen analysis using `cellHTS2` package.

4.1.1 Multiple plate configurations

Although it is good practice to use the same plate configuration for the whole experiment, sometimes this does not work out, and there are different parts of the experiment with different plate configurations. The use of regular expressions in columns *Plate* and *Well* of the plate configuration file allow therefore to specify different configurations within and between assay plates. The two header rows of this file ascertain that all of the plates and wells are covered in the plate configuration file.

Note that in contrast to the `cellHTS` package, in `cellHTS2` package the concept of *batch* is separated from the concept of having multiple plate configurations. So, for example, different replicate of the same plate can be set as to belong to different batches (even though they are required to have the same plate configuration!) - since readouts were performed on different days, or due to the use of different lots of reagents, etc.

Batch information (expressed in terms of integer values giving the batch number: 1, 2, ...) can go into a particular slot called `plateData`. This is expected to be a dataframe) of integer values giving the batch number (1, 2, ...) for each plate and sample. Its first dimension corresponds to the number of individual plates, and its second dimension correspond to the number of columns of each matrix stored in `assayData` slot (the samples). Batch information can be filled in by the user in case s/he wants to take into

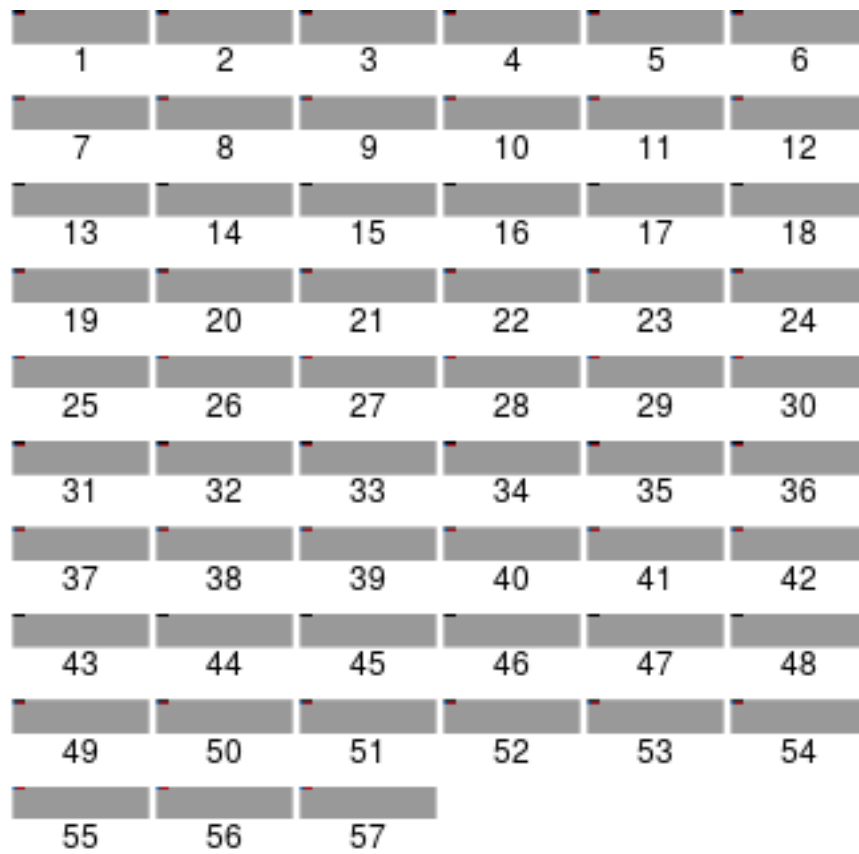


Figure 1: Screen plot of the plate configuration.

account this information in the analysis (for example, see *normalizePlates* function, which allows to adjust the data variance on a per-batch basis). It can either be provided as an optional column in the plate list file, or using the *batch* accessor method.

4.2 Format of the screen log file

The screen log file is a tabulator delimited file with mandatory columns *Plate*, *Well*, *Flag*. If there are multiple samples (replicates or conditions), a column called *Sample* should also be present; a column named *Channel* must also be provided when there are multiple channels. In addition, it can contain arbitrary optional columns. Each row corresponds to one flagged measurement, identified by the plate number (and possible sample number and channel number) and the well identifier (alphanumeric identifier). The type of flag is specified in the column *Flag*. Most commonly, this will have the value “NA”, indicating that the measurement should be discarded and regarded as missing.

For those users that have been using *cellHTS* package and need to migrate their projects to *cellHTS2* package, we explain how this can be smoothly performed in Appendix 10.

5 Normalization, scoring and summarization of replicates

The data normalization, replicates scoring and summarization functions available in *cellHTS2* package work on the data stored in the `assayData` slot of the *cellHTS* object. They create a copy of their input *cellHTS* object, with the data replaced by the transformed values. For a list of the available functions, type `? cellHTS2`.

The preprocessing work-flow of a typical RNAi screen using *cellHTS2* package is the following:

- (a) Per-plate normalization to remove plate and/or edge effects. This can be done using the function *normalizePlates*.
- (b) Scoring of measurements (for example, compute, for each replicate, *z*-scores). This can be done using the function *scoreReplicates*.
- (c) Summarization of replicates (for example, take the median value). This can be done using the *summarizeReplicates*.

Note that this work-flow is suitable for single-channel data. For dual-channel data, further steps are required, as explained in the vignette *Analysis of multi-channel cell-based screens*, accompanying this package.

The function `normalizePlates` can be called to perform per-plate data transformation, normalization and variance adjustment. The normalization is performed in a plate-by-plate fashion and has three components:

Data transformation logarithmic transformation (optional, this can be advisable if the data are on multiplicative scale)

Per-plate normalization location adjustment for possible plate effects and/or possible spatial effects, using a choice of methods that you need to adapt to your data

Variance adjustment plate-specific scale (variance) adjustment (optional)

For more details about this function and available normalization methods, please type `? normalizePlates`. To provide the means to perform the above steps, `normalizePlates` has the arguments `scale`, `log`, `method` and `varianceAdjust`.

The argument `scale` allows to define the scale of the data (“additive” or “multiplicative”), which will then control the subsequent data transformation and normalization steps. Namely, when data are in multiplicative scale, the function allows to perform \log_2 data transformation. For that we need to set the function’s argument `log` to `TRUE`. Log transformation will then change the scale of the data to be “additive”.

Per-plate median normalization is one of the methods available in `normalizePlates` and can be chosen by setting the argument `method="median"`. In this case, plate effects are corrected by dividing (if the current scale of the data is multiplicative) each measurement by the median value across wells annotated as sample, for each plate and replicate. If data are in additive scale, the per-plate median values are subtracted from each plate measurement instead. All of the available normalization methods are described in [Appendix 11](#).

The variance of normalized intensities can be adjusted according to argument `varianceAdjust`, as follows:

- `varianceAdjust="byPlate"`: per plate normalized intensities are divided by the per-plate median absolute deviations (MAD) in “sample” wells. This is done separately for each replicate and channel;

- `varianceAdjust="byBatch"`: using the content of slot `batch`, plates are split according to assay batches and the individual normalized intensities in each group of plates (batch) are divided by the per-"batch of plates" MAD values (calculated based on "sample" wells). This is done separately for each replicate and channel;
- `varianceAdjust="byExperiment"`: each normalized measurement is divided by the overall MAD of normalized values in wells containing "sample". This is done separately for each replicate and channel.

If `varianceAdjust="none"`, no variance adjustment is performed (default).

As explained above, the parameter `method` of `normalizePlates` allows to choose between different types of per-plate normalization methods. Returning to our example data set, we choose to apply *plate median scaling*:

$$x'_{ki} = \frac{x_{ki}}{M_i} \quad \forall k, i \quad (1)$$

$$M_i = \operatorname{median}_{m \in \text{samples}} x_{mi} \quad (2)$$

where x_{ki} is the raw intensity for the k -th well in the i -th replicate file, and x'_{ki} is the corresponding normalized intensity. The median is calculated across the wells annotated as *sample* in the i -th result file. This is achieved by calling:

```
> xn <- normalizePlates(x,
+                       scale="multiplicative",
+                       log=FALSE,
+                       method="median",
+                       varianceAdjust="none")
```

after which we obtain a *cellHTS* object with the normalized intensities stored in the slot `assayData`. In the previous call to `normalizePlates` function, we have chosen not to adjust the data variance (default behaviour of `normalizePlates`). For example, we can use function `compare2cellHTS` provided with the package to check whether these two *cellHTS* objects, `x` and `xn` belong to the same experiment:

```
> compare2cellHTS(x, xn)
```

```
[1] TRUE
```

After normalizing the data, we standardize the values for each replicate experiment using Equation (3). In this equation, $\hat{\mu}$ and $\hat{\sigma}$ are estimators of location and scale of the distribution of x'_{ki} taken across all plates and wells of a given replicate experiment. This function uses robust estimators, namely, median and median absolute deviation (MAD). Moreover, it only considers the wells containing “sample” for estimating $\hat{\mu}$ and $\hat{\sigma}$. The symbol \pm indicates that we allow for either plus or minus sign in Equation (3); the minus sign can be useful in the application to an inhibitor assay, where an effect results in a decrease of the signal and we may want to see this represented by a large score. This is done by calling the `scoreReplicates` function, where arguments `sign` and `method` define the sign and the scoring method to apply (robust z -scores, in this case), respectively:

```
> xsc <- scoreReplicates(xn, sign="-", method="zscore")
```

After data standardization, we summarize the replicates, calculating a single score for each gene. This is performed using the `summarizeReplicates` function, where we use its argument `summary` to select the summary to apply. One option is `rms`, which corresponds to take the root mean square of the replicates values, and is shown in Equation (4). The chosen summary is taken over all the n_{rep_k} replicates of probe k .

$$z_{ki} = \pm \frac{x'_{ki} - \hat{\mu}}{\hat{\sigma}} \quad (3)$$

$$z_k = \sqrt{\frac{1}{n_{\text{rep}_k}} \sum_{r=1}^{n_{\text{rep}_k}} z_{kr}^2}. \quad (4)$$

Depending on the intended stringency of the analysis, other plausible choices of summary function between replicates are the minimum, the maximum, the mean or the median. In the first case, the analysis would be particularly conservative: all replicate values have to be high in order for z_k to be high. For the cases where both sides of the distribution of z -score values are of interest, alternative summary options for the replicates are to select the value closest to zero (conservative approach) by setting `summary="closestToZero"` or the value furthest from zero (`summary="furthestFromZero"`). In order to compare our results with those obtained in the paper of Boutros *et al.* [2], we choose to consider the mean as a summary:

```
> xsc <- summarizeReplicates(xsc, summary="mean")
```

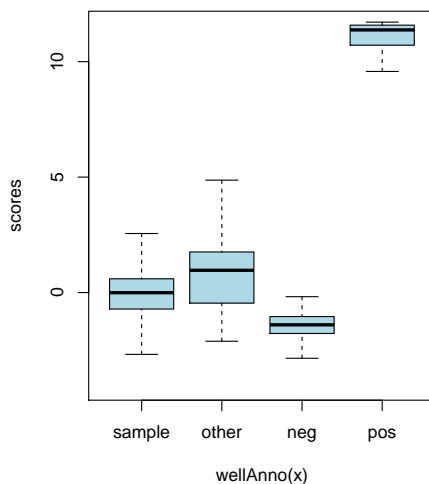


Figure 2: Boxplots of z -scores for the different types of probes.

after which we obtain a *cellHTS* object with the resulting single z -score value per probe stored in `assayData` slot.

Boxplots of the z -scores for the different types of probes are shown in Figure 2.

```
> scores <- Data(xsc)
> ylim <- quantile(scores, c(0.001, 0.999), na.rm=TRUE)
> boxplot(scores ~ wellAnno(x), col="lightblue", outline=FALSE,
+         ylim=ylim)
```

In the *cellHTS2* package, we provide a further transformation of the z -score values to obtain the so-called *calls*. This involves applying a sigmoidal transformation to the z -score values, with parameters z_0 and λ (> 0), given by:

$$y_k = \frac{1}{1 + e^{-\lambda(z_k - z_0)}} \quad (5)$$

This transformation maps the z -score values to the interval $[0, 1]$ and is intended to expand the scale of z -scores with intermediate values and shrink the ones showing extreme values, therefore making the difference between

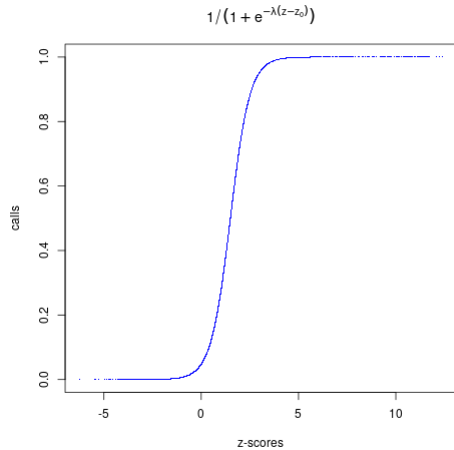


Figure 3: A sigmoidal transformation that can be used for obtaining call values.

intermediate phenotypes larger. The parameter z_0 defines the centre of the sigmoidal transformation, while λ controls the smoothness of the transformation.

This transformation can be done by calling the function `scores2calls`, as shown in Figure 3.

```
> y <- scores2calls(xsc, z0=1.5, lambda=2)
> plot(Data(xsc), Data(y), col="blue", pch=".",
+       xlab="z-scores", ylab="calls",
+       main=expression(1/(1+e^{-lambda *(z-z[0])})))
```

However, for the purpose of the present analysis, we will consider the z -score values instead of the call values.

6 Probe annotation

Up to now, the assayed genes have been identified solely by the identifiers of the plate and the well that contains the probe for them. The *annotation file* contains additional annotation, such as the probe sequence, references to the probe sequence in public databases, the gene name, gene ontology annotation, and so forth. Mandatory columns of the annotation file are *Plate*, *Well*, and *GeneID*, and it has one row for each well. The content of

Plate	Well	HFAID	GeneID
1	A03	HFA00274	CG11371
1	A04	HFA00646	CG31671
1	A05	HFA00307	CG11376
1	A06	HFA00324	CG11723
...

Table 5: Selected lines from the example gene ID file `GeneIDs_Dm_HFA_1.1.txt`.

the *GeneID* column will be species- or project-specific. The first 5 lines of the example file are shown in Table 5, where we have associated each probe with CG-identifiers for the genes of *Drosophila melanogaster*.

```
> xsc <- annotate(xsc, geneIDFile="GeneIDs_Dm_HFA_1.1.txt",
+               path=dataPath)
```

An optional column named *GeneSymbol* can be included in the *annotation file*, and its content will be displayed by the tooltips added to the plate plots and screen-wide plot in the HTML quality report (see Section 7).

6.1 Adding additional annotation from public databases

For the analysis of the RNAi screening results, we usually want to consider gene annotation information such as Gene Ontology, chromosomal location, gene function summaries, homology. The package *biomaRt* can be used to obtain such annotation from public databases [4]. However, there are also numerous alternative methods to annotate a list of gene identifiers with public annotation – pick your favourite one.

This section demonstrates how to do it with the package *biomaRt*. It is optional, you can move on to Section 7 if you do not have the *biomaRt* package or do not want to use it. If you do skip this section, then for the purpose of this vignette, please load a cached version of the gene annotation:

```
> data("bdgpbioMart")
> fData(xsc) <- bdgpbioMart
> fvarMetadata(xsc)[names(bdgpbioMart), "labelDescription"] <-
+   sapply(names(bdgpbioMart),
+         function(i) sub("_", " ", i))
+ )
```

6.1.1 Installation

The installation of the *biomaRt* package can be a little bit tricky, since it relies on the two packages *RCurl* and *XML*, which in turn rely on the presence of the system libraries *libcurl* and *libxml2* on your computer. If you are installing the precompiled R packages (for example, this is what most people do on Windows), then you need to make sure that the system libraries on your computer are version-compatible with those on the computer where the R packages were compiled, and that they are found. If you are installing the R packages from source, then you need to make sure that the library header files are available and that the headers as well as the actual library is found by the compiler and linker. Please refer to the *Writing R Extensions* manual and to the FAQ lists on www.r-project.org.

6.1.2 Using biomaRt to annotate the target genes online

In the remainder of this section, we will demonstrate how to obtain the dataframe `bdgpbioMart` by querying the online webservice *BioMart* and through it the Ensembl genome annotation database [1].

```
> rnwPath <- system.file("doc/Rnw", package="cellHTS2")
> setwd(rnwPath)
> system(sprintf("cp biomart.tex %s", workPath))
> setwd(workPath)

> require("cellHTS2")
> library("biomaRt")
```

By default, the *biomaRt* package will query the webservice at <http://www.ebi.ac.uk/biomart/martservice>. Let us check which BioMart databases it covers:

```
> listMarts()

      biomart
1      ensembl
2          snp
3          vega
4          msd
5          htgt
6      QTL_MART
7  ENSEMBL_MART_ENSEMBL
```

8 ENSEMBL_MART_SNP
 9 GRAMENE_MARKER_29
 10 GRAMENE_MAP_29
 11 REACTOME
 12 wormbase_current
 13 dicty
 14 rgd__mart
 15 ipi_rat__mart
 16 SSLP__mart
 17 g4public
 18 pride
 19 uniprot_mart
 20 ensembl_expressionmart_48
 21 biomartDB
 22 Eurexpress Biomart
 23 pepseekerGOLD_mart06
 24 Pancreatic_Expression

version

1 ENSEMBL 53 GENES (SANGER UK)
 2 ENSEMBL 53 VARIATION (SANGER UK)
 3 VEGA 34 (SANGER UK)
 4 MSD PROTOTYPE (EBI UK)
 5 HIGH THROUGHPUT GENE TARGETING AND TRAPPING (SANGER UK)
 6 GRAMENE 29 QTL DB (CSHL US)
 7 GRAMENE 29 GENES (CSHL US)
 8 GRAMENE 29 SNPs (CSHL US)
 9 GRAMENE 29 MARKERS (CSHL US)
 10 GRAMENE 29 MAPPINGS (CSHL US)
 11 REACTOME (CSHL US)
 12 WORMBASE (CSHL US)
 13 DICTYBASE (NORTHWESTERN US)
 14 RGD GENES (MCW US)
 15 RGD IPI MART (MCW US)
 16 RGD MICROSATELLITE MARKERS (MCW US)
 17 HGNC (EBI UK)
 18 PRIDE (EBI UK)
 19 UNIPROT (EBI UK)
 20 EURATMART (EBI UK)
 21 PARAMECIUM GENOME (CNRS FRANCE)
 22 EUREXPRESS (MRC EDINBURGH UK)

In this example, we use the Ensembl database [1], from which we select the *D. melanogaster* dataset.

```
> mart <- useMart("ensembl",  
+               dataset="dmelanogaster_gene_ensembl")
```

We can query the available gene attributes and filters for the selected dataset using the following functions.

```
> attrs <- listAttributes(mart)  
> filts <- listFilters(mart)
```

In the BioMart system [10], a *filter* is a property that can be used to select a gene or a set of genes (like the “where” clause in an SQL query), and an *attribute* is a property that can be queried (like the “select” clause in an SQL query). We use the *getBM* function of the package *biomaRt* to obtain the gene annotation from Ensembl.

```
> myGetBM <- function(att)  
+   getBM(attributes=c("flybasecgid_gene", att),  
+           filter="flybasecgid_gene",  
+           values=unique(geneAnno(xsc)), mart=mart)
```

For performance reasons, we split up our query in three subqueries, which corresponds to different areas in the BioMart schema, and then assemble the results together in R. Alternatively, it would also be possible to submit a single query for all of the attributes, but then the result table will be enormous due to the 1:many mapping especially from gene ID to GO categories [8].

```
> bm1 <- myGetBM(c("chromosome_name", "start_position",  
+                 "end_position", "description"))  
> bm2 <- myGetBM(c("flybasename_gene", "flybase_gene_id"))  
> bm3 = myGetBM(c("go_biological_process_id",  
+                 "go_biological_process_description"))
```

There are only a few CG-identifiers for which we were not able to obtain chromosomal locations:

```
> length(unique(setdiff(geneAnno(xsc), bm1$flybasecgid_gene)))
```

[1] 389

Below, we add the results to the dataframe `featureData` of `xsc`. Since the tables `bm1`, `bm2`, and `bm3` contain zero, one or several rows for each gene ID, but in `featureData` we want exactly one row per gene ID, the function `oneRowPerId` does the somewhat tedious task of reformatting the tables: multiple entries are collapsed into a single comma-separated string, and empty rows are inserted where necessary.

```
> id <- geneAnno(xsc)
> bmAll <- cbind(
+   oneRowPerId(bm1, id),
+   oneRowPerId(bm2, id),
+   oneRowPerId(bm3, id))
> bdgpbioMart <- cbind(fData(xsc), bmAll)
> fData(xsc) <- bdgpbioMart
> fvarMetadata(xsc)[names(bmAll), "labelDescription"] <-
+   sapply(names(bmAll),
+         function(i) gsub("_", " ", i) )
```

7 Report

We have now completed the analysis tasks: the dataset has been read, configured, normalized, scored, and annotated:

```
> xsc

cellHTS (storageMode: lockedEnvironment)
assayData: 21888 features, 1 samples
  element names: Channel 1
phenoData
  sampleNames: 1
  varLabels: replicate assay
  varMetadata: labelDescription channel
featureData
  featureNames: 1 2 ... 21888 (21888 total)
  fvarLabels: plate well ... go_biological_process_description
             (13 total)
  fvarMetadata: labelDescription
experimentData: use 'experimentData(object)'
state:   configured = TRUE
        normalized = TRUE
```

```

        scored = TRUE
        annotated = TRUE
Number of plates: 57
Plate dimension: nrow = 16, ncol = 24
Number of batches: 1
Well annotation: sample other neg pos
        pubMedIds: 14764878

```

We can now save the scored data set to a file.

```
> save(xsc, file=paste(experimentName, ".rda", sep=""))
```

The data set can be loaded again for subsequent analysis, or passed on to others. To produce a comprehensive report, we can call the function *writeReport* again, this time specifying the three *cellHTS* objects as separate function arguments: “raw”, “normalized” and “scored”. We also alter some of the default output settings using the *setSettings* functions (more details are given in section 7.1)

```

> setSettings(list(plateList=list(reproducibility=list(include=TRUE, map=TRUE),
+                               intensities=list(include=TRUE, map=TRUE)),
+             screenSummary=list(scores=list(range=c(-4, 8), map=TRUE))))
> out <- writeReport(raw=x, normalized=xn, scored=xsc,
+                   force=TRUE)

```

and use a web browser to view the resulting report.

```
> if (interactive()) browseURL(out)
```

The report contains a quality report for each plate, and also for the whole screening assays. The per-plate HTML reports display the scatterplot between duplicated plate measurements, the histogram of the normalized signal intensities for each replicate, and plate plots representing, in a false color scale, the normalized values of each replicate, and the standard deviation between replicate measurements at each plate position. It also reports different measures of agreement between replicate measurements, such as the repeatability standard deviation between replicate plates and the Spearman correlation coefficient between duplicates. The per-plate reports also show the dynamic range, calculated as the ratio between the geometric means of the positive and negative controls. These measures can also be obtained independently from *writeReport* function, by using the functions *getMeasureRepAgreement* and *getDynamicRange* provided in *cellHTS2* package. If

different positive controls were specified at the configuration step and when calling *writeReport*, the dynamic range is calculated separately for the distinct positive controls, since different positive controls might have different potencies.

The experiment-wide HTML report presents, for each replicate, the boxplots with raw and normalized intensities for the different plates, and two plots for the controls: one showing the signal from positive and negative controls at each plate, and another plot displaying the distribution of the signal from positive and negative controls, obtained from kernel density estimates. The latter plot further gives the Z' -factor determined for each experiment (replicate) using the negative controls and each different type of positive controls [13], as a measure to quantify the distance between their distributions. This measure can also be obtained by calling the function *getZfactor*.

The experiment-wide report also shows a screen-wide plot with the z -scores in every well position of each plate. If the argument *map* of *writeReport* function is set to `TRUE`, this plot and the plate plots of the per-plate reports contain tooltips (information pop-up boxes) displaying the annotation information at each position within the plates. If the scored *cellHTS* object provided for *writeReport* is not annotated with gene identifiers, the annotation information shown by the tooltips is simply the well identifiers. For an annotated *cellHTS* object, if an optional column called *GeneSymbol* was included in the *annotation file* (see Section 6), and therefore is present in *featureData* slot of the annotated object, then its content is used for the tooltips. Otherwise, the content of column “GeneID” of the *featureData* slot (which can be accessed via *geneAnno*) is considered.

The screen-wide image plot can also be produced separately using the function *imageScreen* given in the *cellHTS2* package. This might be useful if we want to select the best display for our data, namely, the aspect ratio for the plot and/or the range of z -score values to be mapped into the color scale. These can be passed to the function’s arguments *ar* and *zrange*, respectively. For example,

```
> imageScreen(xsc, ar=1, zrange=c(-3,4))
```

It should be noted that the per-plate and per-experiment quality reports are constructed based on the normalized *cellHTS* object, in case it is provided to *writeReport* function. Otherwise, it uses the data of the raw *cellHTS* object. The quality report produced by *writeReport* function has also a link to a file called *topTable.txt* that contains the list of scored probes

ordered by decreasing z -score values, when the final scored *cellHTS* object is provided. This file has one row for each well and plate, and for the present example data set, it has the following columns:

- **plate** plate identifier for each feature;
- **position** gives the position of the well in the plate (runs from 1 to the total number of wells in the plate);
- **well** gives the alphanumeric well identifier for each feature;
- **score** corresponds to the summarized score calculated for each probe (data stored in the scored and summarized object **xsc**);
- **wellAnno** corresponds to the well annotation (as given by the plate configuration file);
- **finalWellAnno** gives the final well annotation for the scored values. It combines the information given in the plate configuration file with the values in **assayData** slot of the scored *cellHTS* object, in order to have into account the wells that have been flagged either by the screen log file, or manually by the user during the analysis. These flagged wells appear with the annotation *flagged*.
- **raw_r1_ch1** and **raw_r2_ch1** contain the raw intensities for replicate 1 and replicate 2, respectively (data stored in the unnormalized *cellHTS* object **x**). 'ch' refers to channel;
- **median_ch1** corresponds to the median of raw measurements across replicates;
- **diff_ch1** gives the difference between replicated raw measurements (only given if the number of replicates is equal to two);
- **average_ch1** corresponds to the average between replicate raw intensities (only given if the number of replicates is higher than two);
- **raw/PlateMedian_r1_ch1** and **raw/PlateMedian_r2_ch1** give the ratio between each raw measurement and the median intensity in each plate for replicate 1 and replicate 2, respectively. The plate median is determined for the raw intensities, using exclusively the wells annotated as "sample".

- `normalized_r1_ch1` and `normalized_r2_ch1` give the normalized intensities for replicate 1 and replicate 2, respectively. This corresponds to the data stored in the normalized *cellHTS* object `xn`.

Additionally, if any of the *cellHTS* objects provided in the argument `cellHTSlist` to *writeReport* has been annotated (as in the present case), it also contains the data given in the content of `featureData` slot of the annotated object. The above file with the list of scored probes can also be obtained without the need to run *writeReport* by using the function *getTopTable* provided in the package.

7.1 Controlling settings

The *writeReport* function is highly customizable in terms of the resulting HTML output. For most of the graphics that get generated the color scheme, the size, the font size and many other features can be controlled individually. This control is available either through session-wide settings using the *setSettings* function, or for each call of the *writeReport* function through the optional `settings` argument. Most of the plots can even be completely suppressed by switching the respective `include` setting to `FALSE`. Please see *?settings* for more details.

7.2 Exporting data to a tab-delimited file

The *cellHTS2* package contains a function called *writeTab* to save the data stored in `assayData` slot of a *cellHTS* object to a tab-delimited file. The rows of the file are sorted by plate and well, and there is one row for each plate and well. When the *cellHTS* object is annotated, the probe information (i.e. the probe identifiers stored in column “GeneID” of the `featureData` slot) is also added.

```
> writeTab(xsc, file="Scores.txt")
```

Since you might be interested in saving other values to a tab delimited file, below we demonstrate how you can create a matrix with the ratio between each raw measurement and the plate median, together with the gene and well annotation, and export it to a tab-delimited file using the function *write.tabdel*³ also provided in the *cellHTS2* package.

³This function is a wrapper of the function *write.table*, whereby you just need to specify the name of the data object and the file

```

> # determine the ratio between each well and the plate median
> y <- array(as.numeric(NA), dim=dim(Data(x)))
> nrWell <- prod(pdims(x))
> nrPlate <- max(plate(x))
> for(p in 1:nrPlate)
+ {
+   j <- (1:nrWell)+nrWell*(p-1)
+   samples <- wellAnno(x)[j]=="sample"
+   y[j, , ] <- apply(Data(x)[j, , ], drop=FALSE, 2:3,
+                     function(w) w/median(w[samples],
+                                           na.rm=TRUE))
+ }
> y <- signif(y, 3)
> out <- y[, , 1]
> out <- cbind(fData(xsc), out)
> names(out) = c(names(fData(xsc)),
+ sprintf("Well/Median_r%d_ch%d", rep(1:dim(y)[2], dim(y)[3]),
+ rep(1:dim(y)[3], each=dim(y)[2])))
> write.table(out, file="WellMedianRatio.txt")

```

At this point we are finished with the basic analysis of the screen. As one example for how one could continue to further mine the screen results for biologically relevant patterns, we demonstrate an application of category analysis.

8 Category analysis

We would like to see whether there are Gene Ontology categories [8] overrepresented among the probes with a high score. For this we use the category analysis from Robert Gentleman's *Category* package [5]. Similar analyses could be done for other categorizations, for example chromosome location, pathway membership, or categorical phenotypes from other studies.

```

> library("Category")

```

Now we can create the category matrix. Conceptually, this a matrix with one column for each probe and one row for each category. The matrix element [i,j] is 1 if probe j belongs to the j-th category, and 0 if not.

```

> obsolete <- c("GO:0005489", "GO:0015997", "GO:0045034", "GO:0005660",
+              "GO:0006118", "GO:0006512", "GO:0045045", "GO:0006125",
+              "GO:0043072", "GO:0006100", "GO:0048740")

```

Some distractions are the GO terms GO:0005489, GO:0015997, GO:0045034, GO:0005660, GO:0006118, GO:0006512, GO:0045045, GO:0006125, GO:0043072, GO:0006100, GO:0048740, which are annotated to some of the genes, but are obsolete.

```
> scores <- as.vector(Data(xsc))
> names(scores) <- geneAnno(xsc)
> sel <- !is.na(scores) &
+       (!is.na(fData(xsc)$go_biological_process_id))
> goids <- strsplit(fData(xsc)$go_biological_process_id[sel],
+                  ", ")
> goids <- lapply(goids, function(x) x[!(x %in% obsolete)])
> genes <- rep(geneAnno(xsc)[sel], listLen(goids))
> categs <- cateGOdry(genes, unlist(goids, use.names=FALSE))
```

We will select only those categories that contain at least 3 and no more than 1000 genes.

```
> nrMem <- rowSums(categs) # number of genes per category
> remGO <- which(nrMem < 3 | nrMem > 1000)
> categs <- categs[-remGO, , drop=FALSE]
> ## see if there are genes that don't belong to any category
> ## after applying the filter
> nrMem <- rowSums(t(categs))
> rem <- which(nrMem==0)
> if(length(rem)!=0) categs <- categs[, -rem, drop=FALSE]
```

As the statistic for the category analysis we use the z -score. First, we need to select the subset of genes that actually have GO annotation:

```
> stats <- scores[ sel & (names(scores) %in% colnames(categs)) ]
```

There are some replicated probes in `stats`. We will handle this by taking the maximum value between replicate probes (non-conservative approach):

```
> ## handle duplicated genes in stats:
> isDup <- duplicated(names(stats))
> table(isDup)
```

```
isDup
FALSE TRUE
6545  917
```

```

> dupNames <- names(stats)[isDup]
> sp <- stats[names(stats) %in% dupNames]
> sp <- split(sp, names(sp))
> table(sapply(sp, length))

```

```

      2  3  4  5  6  8  9 12
505 120 31 12  1  1  1  1

```

```

> aux <- stats[!isDup]
> aux[names(sp)] <- sapply(sp, max)
> stats <- aux
> rm(aux)

```

Before calling the category summary functions, we need to order our statistic vector according to the names of the columns of the category matrix.

```

> m <- match(colnames(categs), names(stats))
> stats <- stats[m]
> stopifnot(colnames(categs)==names(stats))

```

Finally, we are ready to call the category summary functions:

```

> acMean <- applyByCategory(stats, categs)
> acTtest <- applyByCategory(stats, categs,
+                             FUN=function(v)
+                             t.test(v, stats)$p.value)
> acNum <- applyByCategory(stats, categs, FUN=length)
> isEnriched <- (acTtest<=1e-3) & (acMean>0.5)

```

A volcano plot of the $-\log_{10}$ of the p -value `acTtest` versus the per category mean z -score `acMean` is shown in Figure 4. For a given category, the p -value is calculated from the t -test against the null hypothesis that there is no difference between the mean z -score of all probes and the mean z -score of the probes in that category. To select the enriched categories (`isEnriched`), we considered a significance level of 0.1% for the t -test, and a per category mean z -score greater than 0.5. This led to the 23 categories marked in red in Figure 4 are listed in Table 6.

We have recently added an experimental feature to the `cellHTS2` package in order to integrate such category analyses into the HTML report framework. It leverages functionality from the `GSEABase` package, which was specifically designed to deal with gene sets. The feature is exemplified using the KEGG pathway information available through the `KEGG.db` package.

n	z_{mean}	p	GOID	Ontology	description
338	1.1	5.2e-15	GO:0006412	BP	translation
361	1.1	4.1e-14	GO:0043043	BP	peptide biosynthetic process
368	1	5.7e-14	GO:0006518	BP	peptide metabolic process
373	1	2e-13	GO:0043604	BP	amide biosynthetic process
398	0.94	2.3e-12	GO:0043603	BP	cellular amide metabolic process
721	0.57	1.9e-08	GO:1901566	BP	organonitrogen compound biosynthetic process
834	0.5	1.1e-05	GO:0010468	BP	regulation of gene expression
185	0.72	8.8e-05	GO:0006397	BP	mRNA processing
143	0.74	0.00015	GO:0000375	BP	RNA splicing, via transesterification reactions
143	0.74	0.00015	GO:0000377	BP	RNA splicing, via transesterification reactions with bulged adenosine as nucleophile
143	0.74	0.00015	GO:0000398	BP	mRNA splicing, via spliceosome
148	0.71	0.00016	GO:0008380	BP	RNA splicing
437	0.54	0.00023	GO:0003006	BP	developmental process involved in reproduction
262	0.57	0.00024	GO:0006396	BP	RNA processing
485	0.53	0.00025	GO:0060429	BP	epithelium development
227	0.63	0.00027	GO:0016071	BP	mRNA metabolic process
6	0.99	0.00036	GO:0008335	BP	female germline ring canal stabilization
178	0.74	0.00038	GO:0006911	BP	phagocytosis, engulfment
178	0.74	0.00038	GO:0010324	BP	membrane invagination
178	0.74	0.00038	GO:0099024	BP	plasma membrane invagination
193	0.7	0.00041	GO:0006909	BP	phagocytosis
222	0.63	0.00059	GO:0061024	BP	membrane organization
6	0.6	0.00063	GO:0007561	BP	imaginal disc eversion

Table 6: Top 23 Gene Ontology categories with respect to z -score.

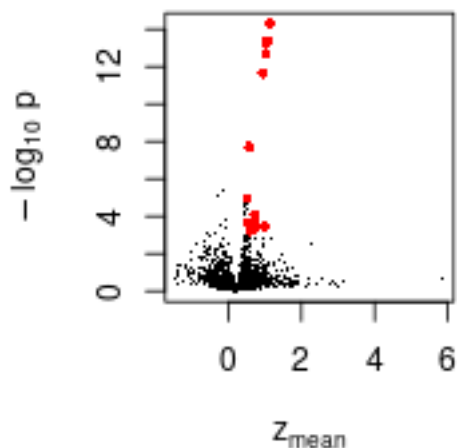


Figure 4: Volcano plot of the t -test p -values and the mean z -values of the category analysis for Gene Ontology categories. The top categories are shown in red.

First we have to create an object of class *GeneSetCollection* which represents the mapping of genes to KEGG pathways. The geneIDs in the gene set collection are supposed to map to the geneIDs of the assay scores. From the *KEGG.db* package we get a list of pathway mapping to FlyBase identifiers.

```
> library(KEGG.db)
> library(GSEABase)
> kegg <- as.list(KEGGPATHID2EXTID)
> kegg <- kegg[grep("dme", names(kegg))]
> gsc <-
+   GeneSetCollection(mapply(function(keggId, geneId)
+     GeneSet(unique(geneId),
+               geneIdType=EntrezIdentifier(),
+               collectionType=KEGGCollection(keggId),
+               setName=keggId),
+     names(kegg), kegg))
```

In a subsequent step we want to filter out all the assay scores without proper annotation (e.g., control wells) and also restrict our collection of gene sets to those containing at least 5 genes. The gene annotations of our assay are the old CG identifiers, but we have retrieved a mapping to proper FlyBase identifiers in the previous *biomaRt* step.

```

> scores <- as.vector(Data(xsc))
> names(scores) <- fData(xsc)$flybase_gene_id
> sel <- !is.na(scores) & !is.na(names(scores))
> scores <- scores[sel]
> ## Only consider set with more than 5 genes
> gsc <- gsc[sapply(gsc, length) > 5]
> gsNames <- mget(gsub("dme", "", names(gsc)), KEGGPATHID2NAME)
> ann <- data.frame(Name=I(unlist(gsNames)))
> rownames(ann) <- names(gsc)

```

Finally, we need to bundle up the assay scores and the gene set collection and define one or several summary statistics to show in the HTML report. Each of these statistics is supposed to be created by a separate function, which takes one mandatory and one optional argument. The mandatory argument is simply a vector of assay scores for the respective gene set. The optional second argument is a numeric vector of all available scores. This setup allows to compute simple statistics like means or medians, but also more complicated test statistics, for instance a *t*-test. The container for this information is an object of class *gseaModule* and there is a convenient constructor which does most of the job.

In the following example, we define four test statistics (mean, *p*-value of a *t*-test, its *t*-statistic and the number of genes in the gene set) and pass them to the constructor as a list. We also add a data.frame with further gene set annotation (in this case more human readable names for the pathways) and the vector of assays scores. Note that if no explicit vector is given, the scores of the *cellHTS* object are used. In our case their identifiers don't match the gene sets FlyBase identifiers, and we also wanted to do a couple of extra filtering steps.

```

> gmod <- gseaModule(gsc, list("Mean"=function(x,...) mean(x),
+                             "P-value"=function(x,y) t.test(x, y)$p.value,
+                             "T-statistic"=function(x,y) t.test(x, y)$statistic,
+                             "NrGenes"=function(x,...) length(x)),
+                          annotation=ann, scores=scores)

```

We can now re-write the *cellHTS* report using the familiar `RfunctionwriteReport` function with the addition `gseaModule` argument.

```

> ## Now load the cellHTS objects and rewrite the report
> out <- writeReport(raw=x, normalized=xn, scored=xsc,
+                   force=TRUE,

```

```
+             outdir=tempdir(), gseaModule=gmod)
> if (interactive()) browseURL(out)
```

9 Comparison with the results previously reported

In this section we compare the current results obtained using *cellHTS* package, with the ones previously reported in Boutros *et al.* [2]. The file “Analysis2003.txt” in the same directory as the input data files, i.e. in KcViab directory of the *cellHTS* package. First, We will load this file:

```
> data2003 <- read.table(file.path(dataPath, "Analysis2003.txt"),
+             header=TRUE, as.is=TRUE, sep="\t")
```

The file contains the columns Plate, Position, Score, Well, HFAID, GeneID. The scored values in the Scores column will be compared with the ones obtained in our analysis. For that, I will start by adding to `data2003`, a column with the corresponding *z*-score values calculated using the *cellHTS* package.

```
> i <- data2003$Position + 384*(data2003$Plate-1)
> data2003$ourScore <- as.vector(Data(xsc))[i]
```

Figure 5 shows the scatterplot between Boutros *et al.*’s scores and our scores in each of the 384-well plates. The results between the two analyses are very similar, except for two minor details: use of robust estimators of location and spread (median and MAD instead of mean and standard deviation), and estimation of MAD over the whole experiment instead of plate-by-plate. In fact, Figure 5 evidenciates how the scored values exactly agree up to an offset (mean versus median) and scale (standard deviation versus MAD).

10 Appendix: How to convert *cellHTS* to *cellHTS2* configuration files

We advise the users of *cellHTS* package to start using the improved package *cellHTS2* described herein, since the latter provides better functionality for working with multi-channel screens and multiple screens.

To facilitate this transition and help users to migrate their *cellHTS*-specific projects to *cellHTS2*, we provide in this package a function that converts the old S3 `cellHTS` object associated with *cellHTS* package into one or several S4 `cellHTS` defined with the *cellHTS2* package. This function is called *convertOldCellHTS*.

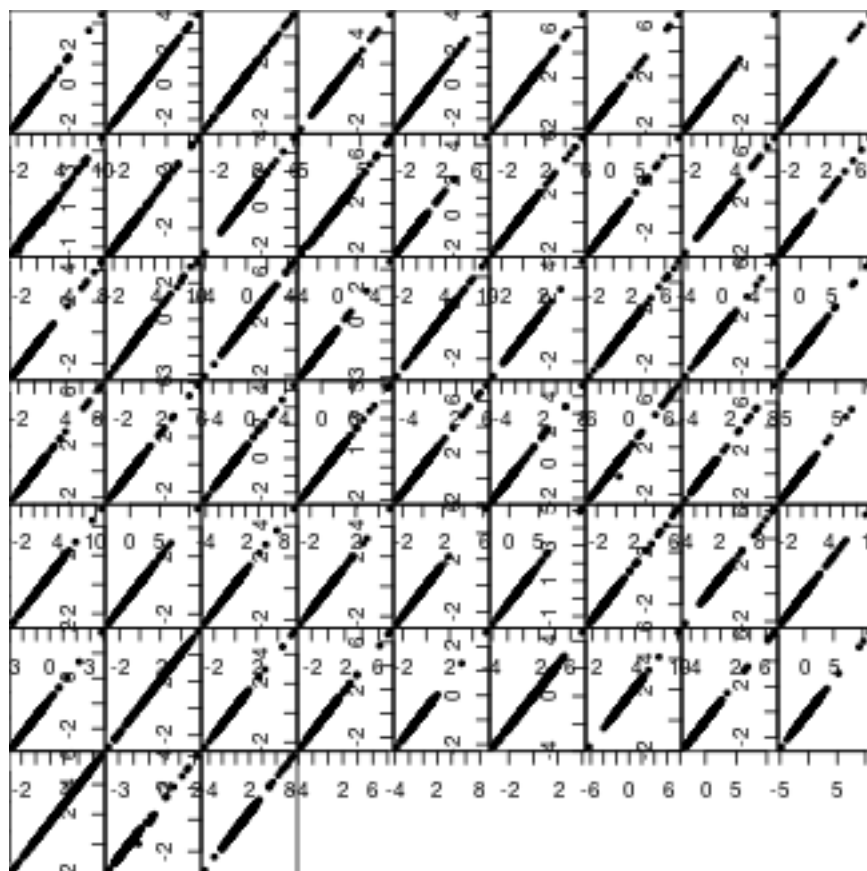


Figure 5: Scored values obtained in the paper of Boutros *et al.* against the scored values calculated herein. Each panel corresponds to one 384-well plate. Axis labels are not pretty - they overlap with neighboring panels due to space constraints.

However, you might want to migrate an existing project from start, i. e., redo all the steps starting by reading the intensity files and configuring the screening data. In this case, you need to update the screen log file (if available), the screen description file and the screen configuration file ⁴.

Regarding the screen description file, as mentioned in Section 4, we provide a function that creates a template screen description file that can be edited and modified by the user. Below we exemplify how such file can be created:

```
> out <- templateDescriptionFile("template-Description.txt",
+                               force=TRUE)
> out

[1] "./template-Description.txt"

> readLines(out)

[1] "[Lab description]"
[2] "Experimenter name: <put here the experimenter name>"
[3] "Laboratory: <put here the name of the laboratory where the experiment was conducted>"
[4] "Contact information: <put here contact information for lab and/or experimenter>"
[5] ""
[6] "[Screen description]"
[7] "Screen: <put here the screen name>"
[8] "Title: <put there the single-sentence giving the experiment title>"
[9] "Version: <put here the screen version>"
[10] "Date: <put here the date when the experiment was performed>"
[11] "Screentype: <put here the type of screen>"
[12] "Organism:"
[13] "Celltype:"
[14] "Library:"
[15] "Assay: <put here the name of the assay>"
[16] "Assaytype: <put here the type of assay>"
[17] "Assaydescription: <put here the description of the assay>"
[18] ""
[19] "[Publication description]"
[20] "Publicationtitle:"
```

⁴The expected format of the other input files, namely the raw intensity data files (Section 2) and the annotation file (Section 6) remains unchanged between the two packages.

```

[21] "Reference:"
[22] "PMIDs: <put here the PubMed identifiers of papers relevant to the dataset>"
[23] "URL: <put here the URL for the experiment>"
[24] "License:"
[25] "Abstract: <put here the abstract describing the experiment>"
[26] ""
[27] "[Files]"
[28] "plateList: <put the name of the plate result list file>"
[29] "annotation: <put the name of the screen library annotation file>"
[30] "plateConf: <put the name of the screen plate configuration file>"
[31] "screenLog: <put the name of screen log file, if available>"

```

The format of the screen log file compatible with *cellHTS2* package is shown in Table 4. Compared to the previous format required for *cellHTS* package (Table 8), we note that column *Filename* was replaced by two columns named *Plate* and *Sample*.

In *cellHTS* package, the concept of *batch* is intrinsically related with the plate configuration, since a change in plate configuration along the experiment had to be handled by setting each distinct plate configuration as corresponding to a different batch. Therefore, in *cellHTS* package, the plate configuration file had three mandatory columns named *Batch*, *Well*, *Content*, where the *Batch* column allowed for different plate configurations. The first 28 lines of such file for the example RNAi screen considered in this report is shown in Table 7. Thus, in the old format required for the plate configuration file, we had to have a number of rows equal to the product between the total number of batches and the total number of wells per plate.

In contrast to *cellHTS* package, in *cellHTS2* package the concept of *batch* and multiple plate configurations were made independent (see Section 4.1.1). For *cellHTS2* package, Table 3 shows the required file format, which was discussed in more detail in Section 4.1.

Due to the separation between batch and multiple plate configuration, the column *Batch* was removed, and replaced by the column *Plate*. The other two mandatory columns *Well* and *Content* were kept. Additionally, we now require that this file contains two extra header lines giving the total number of wells and plates (Table 3). There is also an improvement in the file format related with the fact that *Plate* and *Well* columns now allow the use of regular expressions (see Section 4.1 for more specific information), which allows to cover the plate configuration used in the screen with just a few lines. Besides, it allows to specify different configurations within and

Batch	Well	Content
1	A01	other
1	A02	other
1	A03	sample
1	A04	sample
1	A05	sample
1	A06	sample
1	A07	sample
1	A08	sample
1	A09	sample
1	A10	sample
1	A11	sample
1	A12	sample
1	A13	sample
1	A14	sample
1	A15	sample
1	A16	sample
1	A17	sample
1	A18	sample
1	A19	sample
1	A20	sample
1	A21	sample
1	A22	sample
1	A23	sample
1	A24	sample
1	B01	neg
1	B02	pos
1	B03	sample
1	B04	sample
...

Table 7: Selected lines from the example cellHTS package-specific plate configuration file `old-Plateconf.txt`.

Filename	Well	Flag	Comment
FT06-G01.txt	A01	NA	Contamination
FT06-G02.txt	A01	NA	Contamination
FT06-G01.txt	A02	NA	Contamination
...

Table 8: Selected lines from the example cellHTS package-specific screen log file `old-Screenlog.txt`.

between assay plates.

11 Appendix: Normalization methods implemented in *cellHTS2* package

There are two main normalization methods available with *cellHTS2* package: methods based on the use of reference controls, and distribution-based methods. These methods can be applied using the function *normalizePlates*.

11.1 Controls-based normalization

11.1.1 Percent of control

Percent of control (POC) is a preprocessing method that tries to correct for plate-to-plate variability by normalizing each k th compound raw measurements in the i th result file, x_{ki} , relative to the average of within-plate controls. In an antagonist (or inhibition) type assay, it is defined as:

$$x_{ki}^{\text{POC}} = \frac{x_{ki}}{\mu_i^{\text{pos}}} \times 100 \quad (6)$$

where μ_i^{pos} is the average of the measurements on the positive controls in the i th result file (i. e. , for a given plate and replicate).

In *cellHTS2* package, this method can be applied by setting the argument `method="POC"` when calling *normalizePlates* function.

We also provide in the package a normalization method for *normalizePlates* (`method="negatives"`) that consists of scaling the plate measurements by the per-plate median of the intensities on the negative controls ⁵.

⁵If the scale of the data is defined as being additive (i. e. , argument `scale="additive"`, or arguments `scale="multiply"` and `log=TRUE`), measurements are subtracted by the median of per-plate negative controls instead.

11.1.2 Normalized percent inhibition

If *normalizePlates* is called with `method="NPI"`, the method known as normalized percent inhibition (NPI) is applied in a per-plate basis to correct for plate effects. For an antagonist assay, this method divides the difference between each measurement in a given result file i (x_{ki}) and the average of the positive controls on that plate (μ_i^{pos}) by the difference between the averages of the measurements on the positive (μ_i^{pos}) and the negative controls (μ_i^{neg}):

$$x_{ki}^{\text{NPI}} = \frac{\mu_i^{\text{pos}} - x_{ki}}{\mu_i^{\text{pos}} - \mu_i^{\text{neg}}} \quad (7)$$

11.2 Non-controls-based normalization

There are several normalization methods implemented in *cellHTS2* package that make use of the overall distribution of values, instead of relying exclusively on controls. These are described in the following sections.

11.2.1 Z score method

Z score is a simple and widely known normalizing method that is performed in a per-plate basis as follows:

$$x_{ki}^{\text{Z}} = \frac{x_{ki} - \mu_i}{\sigma_i}, \quad (8)$$

where μ_i and σ_i are the mean and standard deviation, respectively, of all measurements within the i th result file (replicated plate). In the Z score method, measurements are re-scaled relative to within-plate variation by subtracting the average of the plate values and dividing this difference by the standard deviation estimated from all measurements of the plate.

In *cellHTS2*, we consider a robust version of this method, where the mean and standard deviation are replaced by the median and the MAD, respectively, calculated at the sample wells. This robust Z score method is performed by calling *normalizePlates* as follows:

```
> xZ <- normalizePlates(x, scale="additive", log=FALSE,  
+                       method="median",  
+                       varianceAdjust="byPlate")
```

11.2.2 Plate median normalization

Plate median normalization involves calculating the relative signal of each well compared to the median of the sample wells in the plate, as shown in Equation (1) and Equation (2). The median is calculated among the m wells containing *sample* (i. e. , for wells that contain genes of interest) in result file i . Plate median normalization can be chosen by setting `method="median"` in `normalizePlates`. When applied to data on additive scale, the plate median normalization involves subtracting the plate measurements by the per-plate median instead.

We also have two variants of the plate median scaling which consist of using as the per-plate scaling factor M_i the per-plate average intensity on sample wells (`method="mean"`) or the midpoint of the shorth of the per-plate distribution of values on sample wells (`method="shorth"`).

The next methods are intended to explicitly correct for *spatial effects* within plates, i. e. , the presence of intensity gradients within the plates. Such signal gradients can be caused by differences in temperature, incubation time or concentration, etc., in different wells across a plate. Typically, these gradients produce repetitive patterns, which make it possible to distinguish them from real actives that should be more or less randomly dispersed for quasi-randomized collections of compounds.

11.2.3 B score method

In the B score method, row and column biases within each plate are explicitly corrected for by fitting a two-way median polish to the raw data in a per-plate fashion [3]:

$$e_{rci} = x_{rci} - \hat{x}_{rci} = x_{rci} - \left(\hat{\mu}_i + \hat{R}_{ri} + \hat{C}_{ci} \right) \quad (9)$$

$$x_{rci}^B = \frac{e_{rci}}{MAD_i} \quad (10)$$

Here, x_{rci} is the measurement value in the r th row and c th column of the plate corresponding to the i th result file, \hat{x}_{rci} is the corresponding fitted value defined as the sum between the estimated average of the replicate plate ($\hat{\mu}_i$), the estimated systematic offset for row r (\hat{R}_{ri}) and the systematic offset for column c (\hat{C}_{ci}) in that replicated plate. In a second step, each of the obtained residual values e_{rci} 's of the i th result file are divided by their median absolute deviation (MAD_i) giving the final B score value – Equation (10).

We implemented a method similar to the B score method described in Malo *et al.* [11] and Brideau *et al.* [3] using the Tukey’s median polish procedure [12] (function *medpolish* of the package *stats*) which fits an additive model to the data according to Equation (10). The Tukey’s median polish algorithm works by alternately removing the row and column medians and continues until the proportional reduction in the sum of absolute residuals is less than ϵ or until the maximum number of iterations has been reached. The B score method can be applied by defining argument `method="Bscore"` in *normalizePlates*. Alternatively, the method can be applied by calling a separate function called *Bscore* provided in the *cellHTS2* package.

In *cellHTS2* package, we provide two additional spatial normalization methods that fit a polynomial surface to the intensities within each assay plate using local regression and that can be performed via *normalizePlates* or *spatialNormalization* functions, although we advise to apply these methods using the former function. The fit can be performed either using the *loess* procedure or the *locfit.robust* function of package *locfit*. In *normalizePlates*, if `method="locfit"`, spatial effects are removed by fitting a bivariate local regression to each plate and replicate, while if `method="loess"`, a loess curve is fitted instead.

12 Appendix: Data transformation

An obvious question is whether to do the statistical analyses on the original intensity scale or on a transformed scale such as the logarithmic one. Many statistical analysis methods, as well as visualizations work better if (to sufficient approximation)

- replicate values are normally distributed,
- the data are evenly distributed along their dynamic range,
- the variance is homogeneous along the dynamic range [9].

Figure 6 compares these properties for untransformed and log-transformed normalized data, showing that the difference is small. Intuitively, this can be explained by the fact that for small x ,

$$\log(1 + x) \approx x$$

and that indeed the range of the untransformed data is mostly not far from 1. Hence, for the data examined here, the choice between original scale and logarithmic scale is one of taste, rather than necessity.

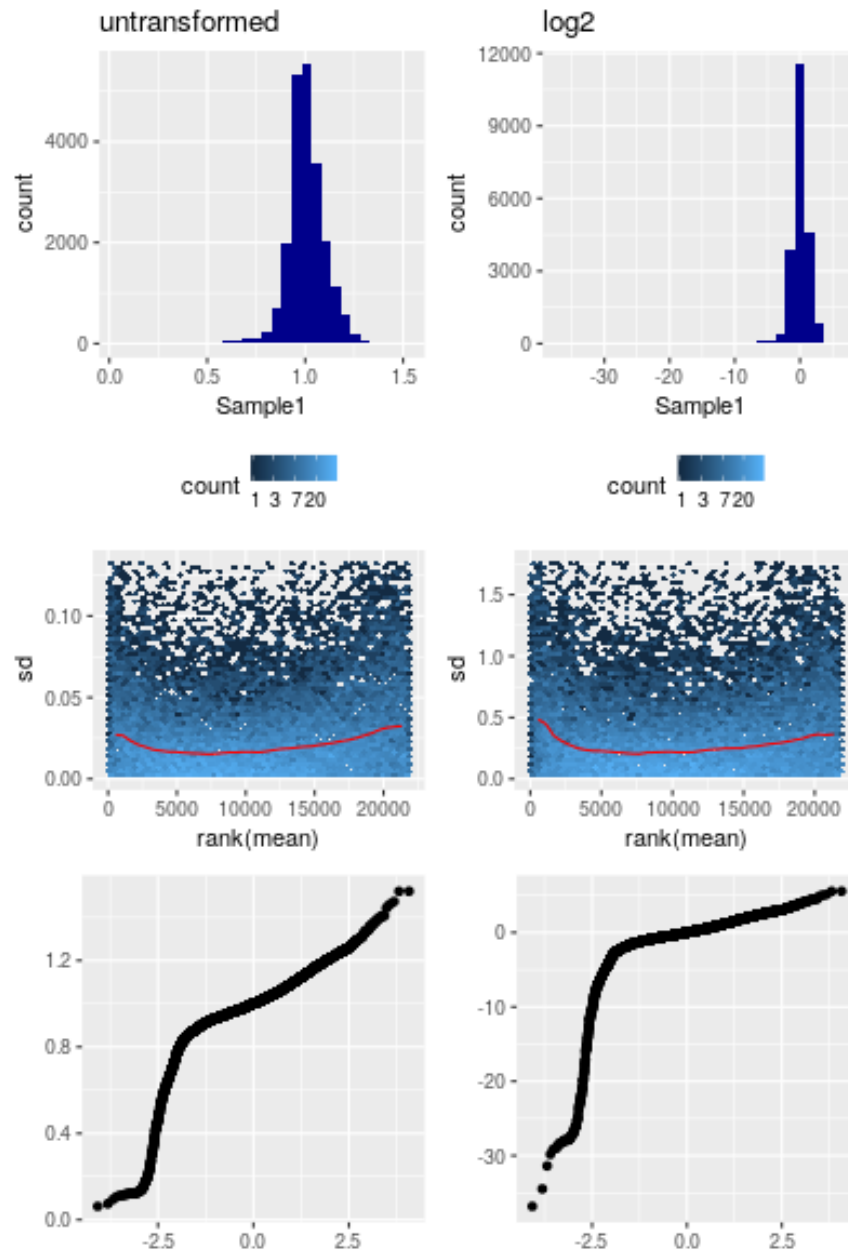


Figure 6: Comparison between untransformed (left) and logarithmically (base 2) transformed (right), normalized data. Upper: histogram of intensity values of replicate 1. Middle: scatterplots of standard deviation versus mean of the two replicates. Bottom: Normal quantile-quantile plots.

13 Session info

References

- [1] E Birney, D Andrews, M Caccamo, Y Chen, L Clarke, G Coates, T Cox, F Cunningham, V Curwen, T Cutts, T Down, R Durbin, X M Fernandez-Suarez, P Flicek, S Graf, M Hammond, J Herrero, K Howe, V Iyer, K Jekosch, A Kahari, A Kasprzyk, D Keefe, F Kokocinski, E Kulesha, D London, I Longden, C Melsopp, P Meidl, B Overduin, A Parker, G Proctor, A Prlic, M Rae, D Rios, S Redmond, M Schuster, I Sealy, S Searle, J Severin, G Slater, D Smedley, J Smith, A Stabenau, J Stalker, S Trevanion, A Ureta-Vidal, J Vogel, S White, C Woodwark, and T J P Hubbard. Ensembl 2006. *Nucleic Acids Res*, 34(Database issue):556–561, 2006. [22](#), [24](#)
- [2] Michael Boutros, Amy A Kiger, Susan Armknecht, Kim Kerr, Marc Hild, Britta Koch, Stefan A Haas, Heidelberg Fly Array Consortium, Renato Paro, and Norbert Perrimon. Genome-wide RNAi analysis of growth and viability in *Drosophila* cells. *Science*, 303(5659):832–835, 2004. [3](#), [18](#), [36](#)
- [3] C Brideau, B Gunter, B Pikounis, and A Liaw. Improved statistical methods for hit selection in high-throughput screening. *J. Biomol. Screen*, 8:634–647, 2003. [43](#), [44](#)
- [4] Steffen Durinck, Yves Moreau, Arek Kasprzyk, Sean Davis, Bart De Moor, Alvis Brazma, and Wolfgang Huber. BioMart and Bioconductor: a powerful link between biological databases and microarray data analysis. *Bioinformatics*, 21(16):3439–3440, Aug 2005. [21](#)
- [5] R. Gentleman. *Category: Category Analysis*, 2006. R package version 1.3.3. [30](#)
- [6] Robert Gentleman. Reproducible research: A bioinformatics case study. *Statistical Applications in Genetics and Molecular Biology*, 3, 2004. [2](#)
- [7] Robert C Gentleman, Vincent J. Carey, Douglas M. Bates, et al. Bioconductor: Open software development for computational biology and bioinformatics. *Genome Biology*, 5:R80, 2004. [6](#)

- [8] M A Harris, J Clark, A Ireland, J Lomax, M Ashburner, R Foulger, K Eilbeck, S Lewis, B Marshall, C Mungall, J Richter, G M Rubin, J A Blake, C Bult, M Dolan, H Drabkin, J T Eppig, D P Hill, L Ni, M Ringwald, R Balakrishnan, J M Cherry, K R Christie, M C Costanzo, S S Dwight, S Engel, D G Fisk, J E Hirschman, E L Hong, R S Nash, A Sethuraman, C L Theesfeld, D Botstein, K Dolinski, B Feierbach, T Berardini, S Mundodi, S Y Rhee, R Apweiler, D Barrell, E Camon, E Dimmer, V Lee, R Chisholm, P Gaudet, W Kibbe, R Kishore, E M Schwarz, P Sternberg, M Gwinn, L Hannick, J Wortman, M Berri-man, V Wood, N de la Cruz, P Tonellato, P Jaiswal, T Seigfried, and R White. The Gene Ontology (GO) database and informatics resource. *Nucleic Acids Res*, 32(Database issue):258–261, 2004. [24](#), [30](#)
- [9] Wolfgang Huber, Anja von Heydebreck, Holger Sültmann, Annemarie Poustka, and Martin Vingron. Variance stabilization applied to microarray data calibration and to the quantification of differential expression. *Bioinformatics*, 18 Suppl. 1:S96–S104, 2002. [44](#)
- [10] Arek Kasprzyk, Damian Keefe, Damian Smedley, Darin London, William Spooner, Craig Melsopp, Martin Hammond, Philippe Rocca-Serra, Tony Cox, and Ewan Birney. EnsMart: a generic system for fast and flexible access to biological data. *Genome Res*, 14(1):160–169, Jan 2004. [24](#)
- [11] Nathalie Malo, James Hanley, Sonia Cerquozzi, Jerry Pelletier, and Robert Nadon. Statistical practice in high-throughput screening data analysis. *Nature Biotechnology*, 24(2):167–175, 2006. [44](#)
- [12] J.W. Tukey. *Exploratory Data Analysis*. Addison-Wesley, Cambridge, MA, 1977. [44](#)
- [13] JH Zhang, TD Chung, and KR Oldenburg. A Simple Statistical Parameter for Use in Evaluation and Validation of High Throughput Screening Assays. *J Biomol Screen*, 4(2):67–73, 1999. [27](#)

```
> toLatex(sessionInfo())
```

- R Under development (unstable) (2019-10-24 r77329),
x86_64-pc-linux-gnu
- Locale: LC_CTYPE=en_US.UTF-8, LC_NUMERIC=C,
LC_TIME=en_US.UTF-8, LC_COLLATE=C, LC_MONETARY=en_US.UTF-8,
LC_MESSAGES=en_US.UTF-8, LC_PAPER=en_US.UTF-8, LC_NAME=C,
LC_ADDRESS=C, LC_TELEPHONE=C, LC_MEASUREMENT=en_US.UTF-8,
LC_IDENTIFICATION=C
- Running under: Ubuntu 18.04.3 LTS
- Matrix products: default
- BLAS: /home/biocbuild/bbs-3.11-bioc/R/lib/libRblas.so
- LAPACK:
/home/biocbuild/bbs-3.11-bioc/R/lib/libRlapack.so
- Base packages: base, datasets, grDevices, graphics, grid, methods,
parallel, stats, stats4, utils
- Other packages: AnnotationDbi 1.49.0, Biobase 2.47.0,
BiocGenerics 0.33.0, Category 2.53.0, GO.db 3.10.0,
GSEABase 1.49.0, IRanges 2.21.0, KEGG.db 3.2.3, Matrix 1.2-17,
RColorBrewer 1.1-2, S4Vectors 0.25.0, XML 3.98-1.20,
annotate 1.65.0, cellHTS2 2.51.0, genefilter 1.69.0, graph 1.65.0,
hwriter 1.3.2, locfit 1.5-9.1, splots 1.53.0, vsn 3.55.0
- Loaded via a namespace (and not attached): BiocManager 1.30.9,
DBI 1.0.0, DEoptimR 1.0-8, MASS 7.3-51.4, R6 2.4.0, RBGL 1.63.0,
RCurl 1.95-4.12, RSQLite 2.1.2, Rcpp 1.0.2, affy 1.65.0, affyio 1.57.0,
assertthat 0.2.1, backports 1.1.5, bit 1.1-14, bit64 0.9-7, bitops 1.0-6,
blob 1.2.0, cluster 2.1.0, colorspace 1.4-1, compiler 4.0.0, crayon 1.3.4,
digest 0.6.22, dplyr 0.8.3, ggplot2 3.2.1, glue 1.3.1, gtable 0.3.0,
hexbin 1.27.3, labeling 0.3, lattice 0.20-38, lazyeval 0.2.2,
limma 3.43.0, magrittr 1.5, memoise 1.1.0, munsell 0.5.0,
mvtnorm 1.0-11, pcaPP 1.9-73, pillar 1.4.2, pkgconfig 2.0.3,
prada 1.63.0, preprocessCore 1.49.0, purrr 0.3.3, rlang 0.4.1,
robustbase 0.93-5, rrcov 1.4-7, scales 1.0.0, splines 4.0.0,
survival 2.44-1.1, tibble 2.1.3, tidyselect 0.2.5, tools 4.0.0, vctrs 0.2.0,
xtable 1.8-4, zeallot 0.1.0, zlibbioc 1.33.0

Table 9: The output of *sessionInfo* on the build system after running this vignette.