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

Michael Boutros, Lígia Brás and Wolfgang Huber

March 1, 2006

Contents

1 Introduction 1

2 Reading the intensity data 2

3 The *cellHTS* class and reports 3

4 Annotating the plate results 4

4.1 Format of the plate configuration file 5

4.1.1 Multiple plate configurations 6

4.2 Format of the screen log file 6

5 Normalization 6

6 Scoring 7

7 Annotation 7

7.1 Adding additional annotation from public databases 9

7.2 Report 12

8 Category analysis 13

1 Introduction

This is a technical report that demonstrates the use of the *cellHTS* package. Its scope is a complete basic analysis of a cell-based high-throughput screen (HTS), from raw intensity readings to an annotated hit list.

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: The first 5 lines from the example plate list file `Platelist.txt`.

This text has been produced as a reproducible document [2]. It contains the actual computer instructions for the method 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 2.2 or greater) together with the package *cellHTS* and a number of other add-on packages. First, we load the required libraries.

```
> library("cellHTS")
> library("xtable")
> library("vsn")
> library("biomaRt")
> library("Category")
> library("GO")
> library("annotate")
```

2 Reading the intensity data

We consider a cell-based screen that was conducted in microtiter plate format. Each of the wells in the plates contains either a gene-specific probe, a control, or it can be empty. Two or perhaps more replicates may have been performed for each plate, and there may be readings from one or more color channels, corresponding to different reporters. Usually, the measurements from each replicate and each color 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*. The first few lines of an example plate list file are shown in Table 1.

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 one comprehensive table that is suitable for subsequent analyses. We now demonstrate the R instructions for this step. We define the path where the input files can be found.

```
> experimentName = "KcViab"
> dataPath = system.file(experimentName, package = "cellHTS")
```

In this example, the input files are in the `KcViab` directory of the *cellHTS* package. Modify this accordingly to read your own data. We show the names of 12 files from this directory

```
> rev(dir(dataPath))[1:12]

 [1] "Screenlog.txt" "Platelist.txt" "Plateconf.txt" "GeneIDs.txt"
 [5] "FT57-G02.txt"  "FT57-G01.txt"  "FT56-G02.txt"  "FT56-G01.txt"
 [9] "FT55-G02.txt"  "FT55-G01.txt"  "FT54-G02.txt"  "FT54-G01.txt"
```

and read the data into the object `x`

```
> x = readPlateData("Platelist.txt", name = experimentName, path = dataPath)

> x

cellHTS object of name 'KcViab'
57 plates with 384 wells, 2 replicates, 1 channel. State:
configured normalized     scored  annotated
     FALSE      FALSE      FALSE      FALSE
```

3 The *cellHTS* class and reports

The basic data structure of the package is the class *cellHTS*. In the previous section, we have created the object `x`, which is an instance of this class. All subsequent analyses, such as normalization, gene selection and annotation, will add their results into this object. Thus, the complete analysis project is contained in this object, and a complete dataset can be shared with others and stored for subsequent computational analyses in the form of such an object. In addition, the 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.

To create a report, use the function *writeReport*. It will create a directory of the name given by `x$name` in the working directory. Alternatively, the argument `outdir` can be specified to direct the output to another directory.

Batch	Pos	Well	Content
1	25	B01	neg
1	26	B02	pos
1	27	B03	sample
1	28	B04	sample
...	...	...	...

Table 2: The first 5 lines from the example plate configuration file `Plateconf.txt`.

Filename	Well	Flag	Comment
FT06-G01.txt	A01	NA	Contamination
FT06-G02.txt	A01	NA	Contamination
FT06-G01.txt	A02	NA	Contamination
FT06-G02.txt	A02	NA	Contamination
FT06-G01.txt	A03	NA	Contamination
...	...	...	...

Table 3: The first 5 lines from the example screen log file `Screenlog.txt`.

```
> writeReport(x)
```

After this function has finished, the index page of the report will be in the file `KcViab/index.html`, and you can view it by directing a web browser to it.

```
> browseURL(file.path(x$name, "index.html"))
```

4 Annotating the plate results

The next step of the analysis is to annotate the measured data with information on controls and to flag invalid measurements. The software expects the information on the controls in a so-called *plate configuration file* (see Section 4.1). This is a tab-delimited file with one row per well.

```
> confFile = file.path(dataPath, "Plateconf.txt")
```

Selected lines of this file are shown in Table 2. Individual measurements can be flagged as invalid in the so-called *screen log file* (see Section 4.2).

```
> logFile = file.path(dataPath, "Screenlog.txt")
```

The first 5 lines of this file are shown in Table 3.

The *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 of the experiments.

```
> descripFile = file.path(dataPath, "Description.txt")
```

We now apply this information to the data object `x`.

```
> x = configure(x, confFile, logFile, descripFile)
```

Moreover, we will also read the screen description file which contains a general description of the screen.

```
> descripFile = file.path(dataPath, "DESCRIPTION.txt")
```

Note that the function *configure*[1] takes `x`, the result from Section 2, as an argument, and we then overwrite `x` with the result of this function.

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 *Batch*, *Pos*, *Well*, *Content*. The *Pos* column runs from 1 to the number of wells in the plate (in the example, 384), and *Well* is the name of the corresponding well in letter-number format (in this case, `A01` to `P24`). The *Content* column can contain one of the following: *sample* (for wells that contain genes of interest), *pos* (for positive controls), *neg* (for negative controls), *empty* (for empty wells), and *other* (for anything that does not fit into the four other categories). Note that these annotations 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(x$plateConf$Content)
```

```
   neg  other    pos sample 
     1      2      1    380 
```

[1] More precisely, *configure* is a method for the S3 class *cellHTS*.

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. It is possible to specify multiple plate configurations simply by appending them to each other in the plate configuration file, and marking them with different numbers in the column *Batch*.

Note that replicated experiments per plate have to use the same plate configuration.

4.2 Format of the screen log file

The screen log file is a tabulator delimited file with mandatory columns *Filename*, *Well*, *Flag*. In addition, it can contain arbitrary optional columns. Each row corresponds to one flagged measurement, identified by the filename and the well 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.

5 Normalization

The function *normalizePlateMedian* adjusts for plate effects by dividing each value in each plate by the median of values in the plate:

$$x'_{ki} = \frac{x_{ki}}{M_i} \qquad \forall k, i \tag{1}$$

$$M_i = \underset{m \in \text{samples}}{\text{median}} \; x_{mi} \tag{2}$$

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

```
> x = normalizePlateMedian(x)
```

after which the normalized intensities are stored in the slot `x$xnorm`. This is an array of the same size as `x$xraw`.

It is easy to define alternative normalization methods, for example, to adjust for additional experimental biases besides the plate effect.

6 Scoring

We can now score the genes. For this, we summarize the replicate values for each gene and calculate the z-score:

$$
\begin{aligned}
y_k &= \max_i x'_{ki} && (3)\\
z_k &= -\frac{y_k - \hat{\mu}}{\hat{\sigma}}. && (4)
\end{aligned}
$$

Here, the summary is taken over all replicates i for gene k. In the case of the example data, we are looking at an inhibitor assay, where an effect results in a decrease of the signal. By using the maximum as the summary function in Equation (3), the analysis is particularly conservative: all replicate values have to be small in order for y_k to be small. Depending on the type of assay and the intended stringency of the analysis, other plausible choices of summary function are the mean and the minimum. $\hat{\mu}$ and $\hat{\sigma}$ are the estimated mean and standard deviation of the summarized values, y_k, annotated as *sample*. We use robust estimates for the mean and the standard deviation, namely, the median for $\hat{\mu}$ and the median absolute deviation (mad) for $\hat{\sigma}$. The minus sign on the right hand side of Equation (4) reflects that we are looking at an inhibitor assay: large positive values of z correspond to a strong effect. For an activator assay, the minus sign is omitted.

```
> x = calcZscore(x, summary = "max", sign = "-")
```

Boxplots of the z-scores for the different types of probes are shown in Figure 1.

```
> ylim = quantile(x$score, c(0.001, 0.999), na.rm = TRUE)
> boxplot(x$score ~ x$wellAnno, col = "lightblue", outline = FALSE,
+     ylim = ylim)
```

7 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 the *GeneID* column will be species- or project-specific. The first 5 lines of

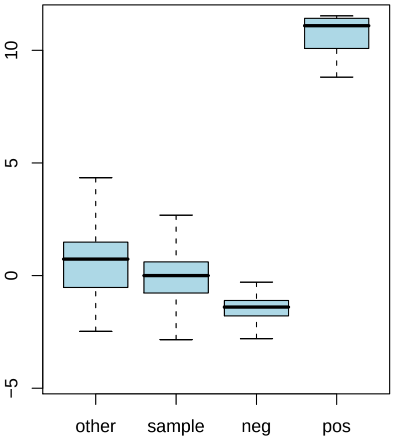

Figure 1: Boxplots of z-scores for the different types of probes.

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

Table 4: The first 5 lines from the example gene ID file `GeneIDs.txt`.

the example file are shown in Table 4, where we have associated each probe with CG-identifiers for the genes of *Drosophila melanogaster*.

```
> geneIDFile = file.path(dataPath, "GeneIDs.txt")
> x = annotate(x, geneIDFile)
```

7.1 Adding additional annotation from public databases

The package *biomaRt* can be used to obtain additional annotation from public databases. First, we load the R package *biomaRt* [1], and check which are the BioMart databases that it currently covers:

```
> library("biomaRt")

> listMarts()

$biomart
[1] "wormbase" "uniprot"  "msd"      "snp"      "vega"     "ensembl"

$version
[1] "WORMBASE CURRENT (CSHL)" "UNIPROT 4-5 (EBI)"
[3] "MSD 4 (EBI)"             "SNP 37 (SANGER)"
[5] "VEGA 37 (SANGER)"        "ENSEMBL 37 (SANGER)"
```

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

```
> mart = useMart("ensembl")

> listDatasets(mart = mart)

                      dataset    version
1    rnorvegicus_gene_ensembl    RGSC3.4
2    scerevisiae_gene_ensembl       SGD1
3       celegans_gene_ensembl     CEL150
4  cintestinalis_gene_ensembl       JGI2
5   ptroglodytes_gene_ensembl    CHIMP1A
6      frubripes_gene_ensembl      FUGU4
7       agambiae_gene_ensembl     AgamP3
8       hsapiens_gene_ensembl     NCBI35
9        ggallus_gene_ensembl    WASHUC1
10   xtropicalis_gene_ensembl       JGI4
```

```
11        drerio_gene_ensembl     ZFISH5
12 tnigroviridis_gene_ensembl TETRAODON7
13      mmulatta_gene_ensembl   MMUL_0_1
14    mdomestica_gene_ensembl    BROADO2
15    amellifera_gene_ensembl    AMEL2.0
16 dmelanogaster_gene_ensembl      BDGP4
17     mmusculus_gene_ensembl    NCBIM34
18       btaurus_gene_ensembl   Btau_2.0
19   cfamiliaris_gene_ensembl    BROADD1
```

```
> mart = useDataset("dmelanogaster_gene_ensembl", mart)
```

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 [4], a *filter* is a property that can be used to select a gene or a set of genes (the "where" clause in an SQL query), and an *attribute* is a property that can be queried (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 = att, filter = "gene_stable_id",
+     values = unique(x$geneAnno$GeneID), mart = mart)
> bm1 = myGetBM(c("gene_stable_id", "chr_name", "chrom_start",
+     "chrom_end", "description"))
> bm2 = myGetBM(c("gene_stable_id", "flybase_name"))
> bm3 = myGetBM(c("gene_stable_id", "go_id", "go_description"))
> unique(setdiff(x$geneAnno$GeneID, bm1$gene_stable_id))
```

```
 [1] NA        "CG33715" "CG33949" "CG32904" "CG33926" "CG33696" "CG33768"
 [8] "CG33769" "CG33770" "CG33936" "CG33937" "CG33630" "CG33950" "CG33653"
[15] "CG33635" "CG33922" "CG33673" "CG33640" "CG33642" "CG33697" "CG33681"
[22] "CG33911" "CG33648" "CG33679" "CG33704" "CR33655" "CG33914" "CG33758"
[29] "CG33757" "CG33800" "CG33919" "CG33627" "CG33752" "CG33775" "CG33792"
[36] "CG33777" "CG33702" "CG33725" "CG33924" "CG33796" "CG33689" "CG33631"
[43] "CG33784" "CG33779" "CG33698" "CG33773" "CR33945" "CG33651" "CR33939"
[50] "CG33639"
```

```
> table(table(bm1$gene_stable_id))
```

```
    1     2     4
12338    24    20
```

```
> table(table(bm2$gene_stable_id))

    1     2     3     4     5     6     7     8     9    10    11    12    13
 9548  1683   562   269   110    66    36    48     8     9     7     2     4
   14    15    16    17    25    26  4096  8192 12288
    1     2     3     2     1     1    18     1     1
```

```
> table(table(bm2$gene_stable_id))

    1     2     3     4     5     6     7     8     9    10    11    12    13
 9548  1683   562   269   110    66    36    48     8     9     7     2     4
   14    15    16    17    25    26  4096  8192 12288
    1     2     3     2     1     1    18     1     1
```

In the code above, although it would be possible to run a single query for all of the attributes, we run three separate queries, in order to avoid enormous blow-up of the result table due to the 1:many mapping especially from gene ID to GO categories. Below, we add the results to the dataframe `x$geneAnno`. Since the tables `bm1`, `bm2`, and `bm3` contain zero, one or several rows for each gene ID, but in `x$geneAnno` we want exactly one row per gene ID, the function *format* 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.

```
> format = function(ids, x) {
+     stopifnot(all(x[, 1] %in% ids))
+     d = lapply(2:ncol(x), function(i) {
+         r = character(length(ids))
+         v = sapply(split(x[, i], x[, 1]), unique)
+         v = sapply(v, paste, collapse = ", ")
+         mt = match(names(v), ids)
+         r[mt] = v
+         r[r == ""] = NA
+         return(I(r))
+     })
+     names(d) = colnames(x)[2:ncol(x)]
+     res = do.call("data.frame", d)
+ }
```

```
> x$geneAnno = cbind(x$geneAnno, format(x$geneAnno$GeneID, bm1),
+     format(x$geneAnno$GeneID, bm2), format(x$geneAnno$GeneID,
+         bm3))
```

7.2 Report

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

```
> x

cellHTS object of name 'KcViab'
57 plates with 384 wells, 2 replicates, 1 channel. State:
configured normalized     scored  annotated
      TRUE       TRUE       TRUE       TRUE
```

We can now save the data set to a file.

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

The dataset can be loaded again for subsequent analysis, or passed on to others. To produce a comprehensive report, we can call the function *writeReport* again,

```
> writeReport(x, force = TRUE, plotPlateArgs = list(xrange = c(0.6,
+     1.4)), imageScreenArgs = list(zrange = c(-2, 6.5), ar = 1))
```

and use a web browser to view the resulting report

```
> browseURL(file.path(x$name, "index.html"))
```

Now, the report contains a quality report for each plate, and also for the whole screening assays. The experiment-wide report presents the Z'-factor determined for each experiment (replicate) using the positive and negative controls, the boxplots with raw and normalized intensities for the different plates, and the screen-wide plot with the z-scores in every well position of each plate.

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 overrepresented among the probes with a high score. For this we use the category analysis from Robert Gentleman's *Category* package. Similar analyses could be done for other categorizations, for example chromosome location, pathway membership, or categorical phenotypes from other studies.

Now we can create the category matrix. 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.

```
> names(x$score) = x$geneAnno$GeneID
> selsc = !is.na(x$score)
> selbm = (bm3$gene_stable_id %in% names(which(selsc))) & (bm3$go_id !=
+     "")
> categs = cellHTS:::cache("categs", cateGOry(bm3$gene_stable_id[selbm],
+     bm3$go_id[selbm]))
```

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

```
> nrMem = listLen(edges(categs))
> categs = subGraph(nodes(categs)[nrMem >= 3 & nrMem <= 1000],
+     categs)
```

As the statistic for the category analysis we use the z-score. After selecting the subset of genes that actually have GO annotation,

```
> stats = x$score[selsc & (names(x$score) %in% nodes(categs))]
```

we are ready to call the category summary functions:

```
> acMean = applyByCategory(stats, categs)
> acTtest = applyByCategory(stats, categs, FUN = function(v) t.test(v,
+     x$score)$p.value)
> acNum = applyByCategory(stats, categs, FUN = length)
> isEnriched = (acTtest <= 0.001) & (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 2. The p-value is calculated from the t-test against the null hypothesis $H_0 : z = 0$. 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 27 categories marked in red in Figure 2 are listed in Table 5.

n	$z_{\mathbf{mean}}$	p	**GOID**	**Ontology**	
842	0.57	4.3e-19	GO:0043234	CC	
119	2.2	5.7e-19	GO:0005840	CC	
184	1.6	2.8e-18	GO:0030529	CC	
486	0.7	4.9e-16	GO:0043228	CC	
486	0.7	4.9e-16	GO:0043232	CC	intracellu
87	1.7	7e-10	GO:0005829	CC	
45	2.7	5e-09	GO:0000502	CC	prot
211	0.67	8.4e-08	GO:0005783	CC	
19	3.8	2e-06	GO:0005838	CC	proteasome reg
24	2.1	0.00016	GO:0005839	CC	proteaso
15	2.3	0.00038	GO:0015934	CC	
294	1.1	3.2e-18	GO:0006412	BP	
336	0.92	3e-17	GO:0009059	BP	
575	0.62	6.6e-17	GO:0044249	BP	
638	0.52	1.1e-14	GO:0009058	BP	
73	0.68	0.00035	GO:0006397	BP	
76	0.66	4e-04	GO:0016071	BP	
4	0.68	0.00058	GO:0007561	BP	
43	0.9	0.00088	GO:0000375	BP	RNA splici
43	0.9	0.00088	GO:0000377	BP	RNA splicing, via transesterification reactions wi
43	0.9	0.00088	GO:0000398	BP	nucle
47	0.82	0.00096	GO:0008380	BP	
129	2.1	1.8e-18	GO:0003735	MF	
305	0.84	2.7e-13	GO:0005198	MF	
24	2.1	0.00016	GO:0004298	MF	
58	0.74	0.00033	GO:0008135	MF	translation
61	0.68	0.00053	GO:0045182	MF	

Table 5: Top 27 Gene Ontology categories with respect to z-score.

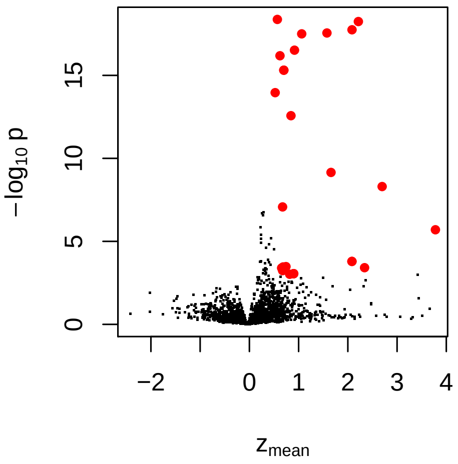

Figure 2: Volcano plot of the t-test p-values and the mean z-values of the category analysis for Gene Ontology categories. The top categories are shown in red.

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 [3].

Figure 3 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.

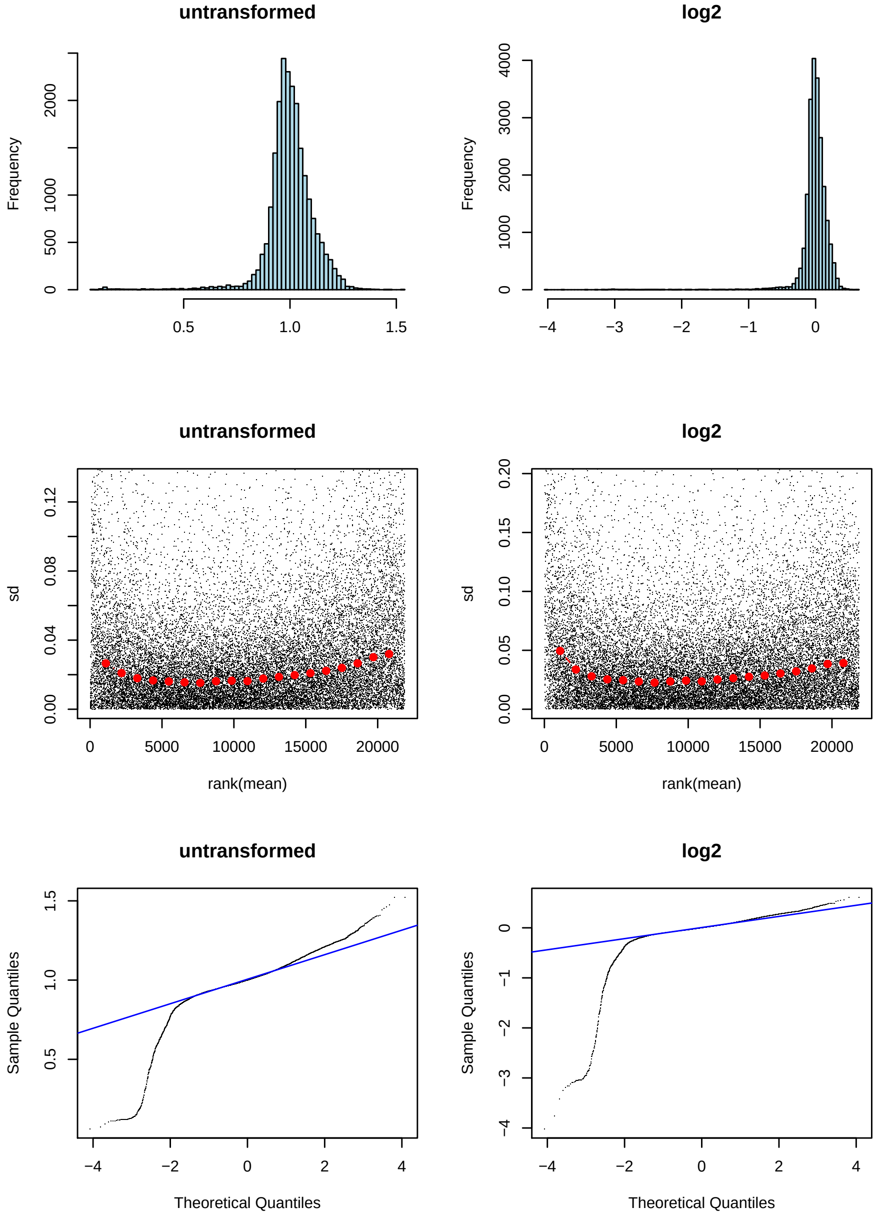

Figure 3: Comparison between untransformed (left) and logarithmically (base 2) transformed (right), normalized data. Upper: histogram of intensity values of replicate 1. Middle: scatterplots of standard deviation versus mean of the two replicates. Bottom: Normal quantile-quantile plots.

```
> par(mfcol = c(3, 2))
> myPlots = function(z, ...) {
+     hist(z[, 1], 100, col = "lightblue", xlab = "", ...)
+     meanSdPlot(z, ylim = c(0, quantile(abs(z[, 2] - z[, 1]),
+         0.95, na.rm = TRUE)), ...)
+     qqnorm(z[, 1], pch = ".", ...)
+     qqline(z[, 1], col = "blue")
+ }
> dv = matrix(x$xnorm, nrow = prod(dim(x$xnorm)[1:2]), ncol = dim(x$xnorm)[3])
> myPlots(dv, main = "untransformed")
> myPlots(log2(dv), main = "log2")
```

References

[1] 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.

[2] Robert Gentleman. Reproducible research: A bioinformatics case study. *Statistical Applications in Genetics and Molecular Biology*, 3, 2004.

[3] 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.

[4] 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.