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 14, 2006

Contents

1 Introduction

This is a technical report that demonstrates the use of the *cellHTS* package. It accompanies the paper *Analysis of cell-based RNAi screens* by Michael Boutros, Ligia Bras and Wolfgang Huber. This report explains 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 [5]. 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 some other add-on 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 *cellhts.Rnw* in the *doc/Rnw* directory of the package, open it in a text editor, run it using the R command *Sweave*, and modify it to your needs.

First, we load the package.

```
> library("cellHTS")
```

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, which is indicative of ATP levels. Although this set of example data corresponds to a single-channel screening assay, the *cellHTS* package can also deal with cases where there are readings from 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.

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`.

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 is one big table 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 = "cellHTS")
```

In this example, the input files are in the `KcViab` directory of the *cellHTS* package. To read your own data, modify `dataPath` to point to the directory where they reside. We show the names of 12 files from our example directory:

```
> dataPath
```

```
[1] "/home/ligia/R/Rpackages/cellHTS/KcViab"
```

```
> 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
```

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

Table 2: Selected lines from the example plate configuration file `Plateconf.txt`.

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.

```
> writeReport(x)
```

It can take a while to run this function, since it writes a large number of graphics files. 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(out, "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.

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
...	...	...	...

Table 3: Selected lines from the example screen log file `Screenlog.txt`.

```
> 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)
```

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

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

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
```

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 corresponding 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. You might want to start by taking the source code of *normalizePlateMedian* as a template.

6 Scoring

We can now score the genes. For this, we calculate the z-score for each probe in each replicate j:

$$z_{kj} = -\frac{x'_{kj} - \hat{\mu}_j}{\hat{\sigma}_j} \tag{3}$$

where x'_{kj} is the normalized intensity for the k-th well in the j-th replicate, $\hat{\mu_j}$ and $\hat{\sigma_j}$ are the estimated mean and standard deviation for the normalized values annotated as *sample* within all the plates of the j-th replicate. We use robust estimates for the mean and the standard deviation, namely, the median for $\hat{\mu_j}$ and the median absolute deviation (mad) for $\hat{\sigma_j}$. The minus sign on the right hand side of Equation (3) reflects that, in the case of the example data, we are looking at an inhibitor assay, where an effect results in a decrease of the signal. For an activator assay, the minus sign is omitted. Thus, in both type of assays, large positive values of z_{kj} will correspond to a strong effect.

To obtain a single z-score value per probe, we take the minimum of the z-scores from the two replicates:

$$z_k = \min_j z_{kj}. \tag{4}$$

The summary is taken over all replicates for probe k. By using the minimun as the summary function in Equation (4), the analysis is particularly conservative: all replicate values have to be high in order for z_k to be high. Depending on the intended stringency of the analysis, other plausible choices of summary function are the mean and the maximum. Since we want to produce the HTML quality reports (see Section 8) using the normalized

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

Table 4: Selected lines from the example gene ID file `GeneIDs.txt`.

and scored replicate values (z_{kj}), we will call again the *normalizePlateMedian*, and choose the option to score the replicates after scaling by the plate median[2]:

```
> x = normalizePlateMedian(x, zscore = "-")
```

Note that this will replace the content in `x$xnorm` by the scored replicate values z_{kj}. Then, using the function below, we summarize the scored replicates, obtaining a single z-score value per probe that will be stored in the slot `x$score`:

```
> x = summarizeZscore(x, summary = "min")
```

The content of `x$score` should always correspond to the final list of scored probes. 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

[2]The *cellHTS* package also provides a function called *calcZscore* that first summarizes the normalized values across replicates, and then calculates a z-score value for each probe. However, in the present example, we prefer to have the normalized and scored replicates in `x$xnorm`, and use them for constructing the quality report.

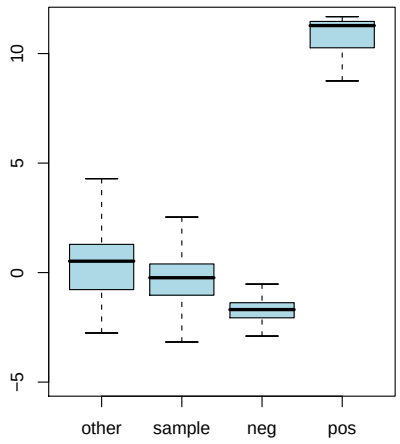

Figure 1: Boxplots of z-scores for the different types of probes.

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 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 [3]. You will need version 1.5.3 or later of the *biomaRt* package. This is currently (March 2006) in the development branch of Bioconductor and will be in the release 1.8.

```
> library("biomaRt")
```

7.1.1 Installation

The installation of the biomaRt can be a little bit tricky, since it relies on the two packages *RCurl* and *XML*, which in turn rely on the system libraries *libcurl* and *libxml2*. 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 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.

Naturally, before facing an extensive installation struggle, you will want to explore the *cellHTS* package and see whether it is worthwhile. We have made available the result of the annotation with *biomaRt*, described below, as a precomputed R object:

```
> data("bdgpbiomart")
> x$geneAnno = bdgpbiomart
```

7.1.2 Using biomaRt to annotate the target genes online

The commands in this section are optional, you can move on to Section 8 if you do not have the *biomaRt* package or do not want to use it. 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].

By default, the *biomaRt* package will query the webservice at http://www.ebi.ac.uk/biomart/martse Let's check which BioMart databases it covers:

```
> 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 use the Ensembl database [1], 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    BROAD02
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 [8], 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("gene_stable_id",
+     att), filter = "gene_stable_id", values = unique(x$geneAnno$GeneID),
+     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 blown up enormously due to the 1:many mapping especially from gene ID to GO categories [6].

```
> bm1 = myGetBM(c("chr_name", "chrom_start", "chrom_end", "description"))
> bm2 = myGetBM(c("flybase_name"))
> bm3 = myGetBM(c("go_id", "go_description"))
```

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

```
> 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"
```

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 *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 = x$geneAnno$GeneID
> bmAll = cbind(oneRowPerId(bm1, id), oneRowPerId(bm2, id), oneRowPerId(bm3,
+     id))
> identical(bdgpbiomart[, 5:11], bmAll)
```

```
[1] TRUE
```

8 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 = ""), compress = TRUE)
```

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(-6,
+     5)), 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 [9], 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. It should be noted that the per-plate and per-experiment quality reports are constructed based on the content of `x$xnorm`, if it is present in the `x` object. Otherwise, it uses the content given in the slot `x$xraw`. Although we have called the former slot, `xnorm`, it can contain already scored replicate data (as in the present case; see Section 6), or, in the case of dual-channel experiments, the ratio between the intensities in two different channels, etc. The main point that we want to highlight is that `x$xnorm` should contain the data that we want to visualize in the HTML quality reports. On the other hand, `x$score` should always contain the final list of scored probes (one value per probe).

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.

9 Category analysis

We would like to see whether there are Gene Ontology categories [6] overrepresented among the probes with a high score. For this we use the category analysis from Robert Gentleman's *Category* package [4]. Similar analyses could be done for other categorizations, for example chromosome location, pathway membership, or categorical phenotypes from other studies.

You will need version 1.3.2 or later of the *Category* package. This is currently (March 2006) in the development branch of Bioconductor and will be in the release 1.8.

```
> library("Category")
> stopifnot(package.version("Category") >= package_version("1.3.2"))
```

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.

```
> remGO = which(regexpr("^GO:", nodes(categs)) > 0)
> nrMem = listLen(edges(categs)[remGO])
> remGO = remGO[nrMem > 1000 | nrMem < 3]
> categs = subGraph(nodes(categs)[-remGO], 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,
+     stats)$p.value)
> acNum = applyByCategory(stats, categs, FUN = length)
> isEnriched = (acTtest <= 0.001) & (acMean > 0.5)
```

n	$z_{\mathbf{mean}}$	p	**GOID**	**Ontology**	**description**
119	2.1	1.8e-18	GO:0005840	CC	ribosome
184	1.4	1.7e-17	GO:0030529	CC	ribonucleoprotein complex
87	1.5	1.6e-09	GO:0005829	CC	cytosol
45	2.6	7.5e-09	GO:0000502	CC	proteasome complex (sensu Eukaryota)
19	3.7	2.4e-06	GO:0005838	CC	proteasome regulatory particle (sensu Eukaryota)
24	1.9	0.00021	GO:0005839	CC	proteasome core complex (sensu Eukaryota)
15	2.2	0.00046	GO:0015934	CC	large ribosomal subunit
294	0.87	5.8e-17	GO:0006412	BP	protein biosynthesis
336	0.71	7.8e-16	GO:0009059	BP	macromolecule biosynthesis
129	1.9	6e-18	GO:0003735	MF	structural constituent of ribosome
305	0.64	3.7e-12	GO:0005198	MF	structural molecule activity
24	1.9	0.00021	GO:0004298	MF	threonine endopeptidase activity
58	0.53	0.00068	GO:0008135	MF	translation factor activity, nucleic acid binding

Table 5: Top 13 Gene Ontology categories with respect to z-score.

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. 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 13 categories marked in red in Figure 2 are listed in Table 5.

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 [7].

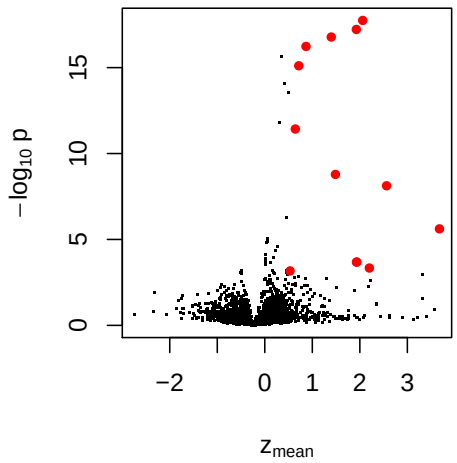

Figure 2: Volcano plot of the t-test p-values and the mean z-values of the category analysis for Gene Ontology categories. The top categories are shown in red.

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.

```
> library("vsn")
> 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")
> xlog = normalizePlateMedian(x, transform = log2, zscore = "-")
```

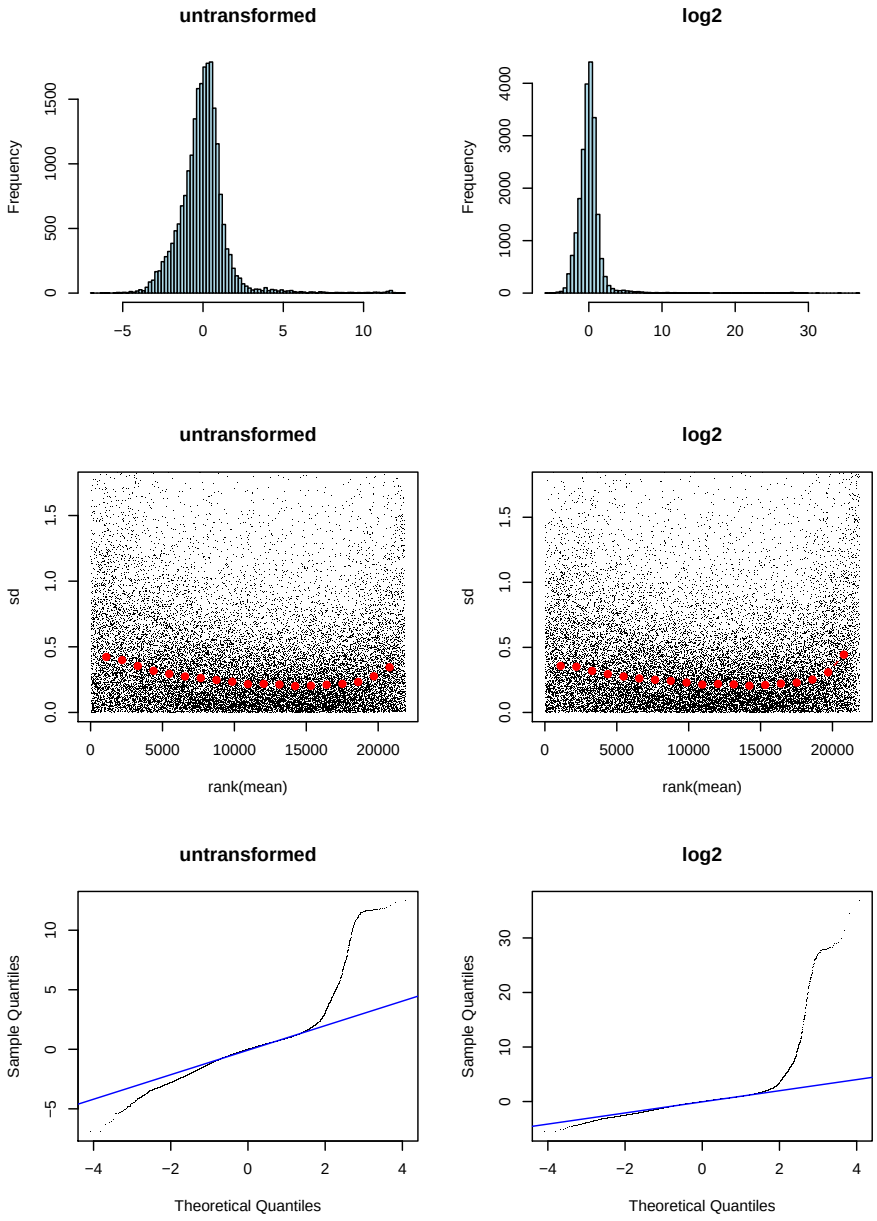

Figure 3: Comparison between untransformed (left) and logarithmically (base 2) transformed (right), normalized data. Upper: histogram of intensity values of replicate 1. Middle: scatterplots of standard deviation versus mean of the two replicates. Bottom: Normal quantile-quantile plots.

```
> dvlog = matrix(xlog$xnorm, nrow = prod(dim(xlog$xnorm)[1:2]),
+     ncol = dim(xlog$xnorm)[3])
> myPlots(dvlog, main = "log2")
```

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, Jan 2006. 10

[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, Feb 2004. 2

[3] 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. 9

[4] R. Gentleman. *Category: Category Analysis*, 2006. R package version 1.3.3. 14

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

[6] 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 Berriman, 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, Jan 2004. 12, 14

[7] 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. 15

[8] 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. 11

[9] 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. 13