Analysis of multi-channel cell-based screens

Lígia Brás, Michael Boutros and Wolfgang Huber

November 1, 2022

Contents

1 Introduction

This techical report is a supplement of the main vignette *End-to-end analysis of cell-based screens: from raw intensity readings to the annotated hit list* that is given as part of the *cellHTS2* package.

The report demonstrates how the *cellHTS2* package can be applied to the documentation and analysis of multi-channel cell-based high-throughput screens (HTS), more specifically, dual-channel experiments. Such experiments are used, for example, to measure the phenotype of a pathway-specific reporter gene against a constitutive signal that can be used for normalization purposes. Typical examples for dual-channel experimental setups are dual-luciferase assays, whereby both a firefly and renilla luciferase are measured in the same well. In principle, multiplex assays can consist of many more than two channels, such as in the case of flow-cytometry readout or other microscopy-based high-content approaches.

We note that in this report we present a simple approach to analyse data from dual-channel experiments, which can be expanded to experiments

with more than two reporters, taking the in-built normalization functions of *cellHTS2* as a template, and employing the extensive statistical modeling capabilities of the R programming language. Moreover, such analyses should be adapted to the biological system and to the question of interest.

This text has been produced as a reproducible document [1], containing the actual computer instructions, given in the language R, to produce all results, including the figures and tables that are shown here. 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. Then, you can simply take the file *twoChannels.Rnw* in the *doc* directory of the package, open it in a text editor, run it using the R command *Sweave*, and modify it according to your needs.

We start by loading the package.

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

2 Assembling the data

Here, we consider a sample data of a dual-channel experiment performed with *D. melanogaster* cells. The screen was conducted in microtiter plate format using a library of double-stranded RNAs (dsRNAs), in duplicates. The example data set corresponds to three 384-well plates. The purpose of the screen is to find signaling components of a given pathway. In the screen, one reporter (assigned to channel 1, and denoted here by R_1) monitors cell growth and viability, while the other reporter (assigned to channel 2 and denoted here by R_2) is indicative of pathway activity.

2.1 Reading the raw intensity files

The set of available result files and the information about them (which plate, which replicate, which channel) is given in the *plate list file*. The first few lines of the plate list file for this data set are shown in Table 1.

Using the function *readPlateData*, we can read the plate list file and all of the intensity files, thereby assembling the data into a single R object that can be used for subsequent analyses. First, we define the path for those files:

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

The input files are in the `DualChannelScreen` directory of the *cellHTS2* package.

Filename	Plate	Replicate	Channel
RA01D1.TXT	1	1	1
RA01D2.TXT	1	2	1
RA02D1.TXT	2	1	1
RA02D2.TXT	2	2	1
RA03D1.TXT	3	1	1
...	...	...	...

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

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

> x

cellHTS (storageMode: lockedEnvironment)
assayData: 1152 features, 2 samples
  element names: Channel 1, Channel 2
phenoData
  sampleNames: 1 2
  varLabels: replicate assay
  varMetadata: labelDescription channel
featureData
  featureNames: 1 2 ... 1152 (1152 total)
  fvarLabels: plate well controlStatus
  fvarMetadata: labelDescription
experimentData: use 'experimentData(object)'
state:  configured = FALSE
        normalized = FALSE
        scored = FALSE
        annotated = FALSE
Number of plates: 3
Plate dimension: nrow = 16, ncol = 24
Number of batches: 1
```

2.2 Annotating the plate results

Next, we annotate the measured data with information on the controls, and flag invalid measurements using the information given in the *plate configuration file* and in the *screen log file*, respectively. Selected lines of these files are shown in Table 2 and Table 3. Morevoer, we also add the information

Wells:	384	
Plates:	3	
Plate	Well	Content
*	*	sample
*	A01	geneA
*	A02	geneB
*	B01	geneC
*	B02	geneD

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

Plate	**Sample**	**Channel**	**Well**	**Flag**	**Comment**
3	1	1	A05	NA	contaminated
3	1	2	A05	NA	contaminated
...	...	...	...	...	...

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

contained in the *screen description file*, which gives a general description of the screen.

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

In this data set, instead of using the default names *pos* and *neg* for positive and negative controls, respectively, we use the name of the gene targeted by the probes in the control wells: *geneA*, *geneB*, *geneC* and *geneD*. This is a more straighforward approach, since not all of these four controls behave as controls for both reporters R_1 and R_2. Moreover, the two positive controls have different strengths: *geneC* is expected to generate a weaker effect than *geneD*. Thus, it is useful to define these controls separately at the configuration step, in order to calculate the quality measures (dynamic range and Z'-factors) specific for each of them in the HTML quality reports or by calling the function *getDynamicRange* and *getZfactor*.

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

```
> table(wellAnno(x))
```

```
sample  genea  geneb  genec  gened
  1140      3      3      3      3
```

3 Data preprocessing

We can take a first look at the data by constructing the HTML quality reports using the *writeReport* function.

As mentioned above, the controls used in the screen are reporter-specific. When calling *writeReport*, we need to specify to the function's arguments `posControls` and `negControls` which are the positive and negative controls for each channel:

```
> ## Define the controls for the different channels:
> negControls <- vector("character", length=dim(Data(x))[3])
> # channel 1 - gene A
> negControls[1] <- "(?i)^geneA$"
> ## case-insensitive and match the empty string at the beginning and
> ## end of a line (to distinguish between "geneA" and "geneAB", for example.
> ## Although it is not a problem for the present well annotation)
>
> # channel 2 - gene A and geneB
> negControls[2] <- "(?i)^geneA$|^geneB$"
> posControls <- vector("character", length=dim(Data(x))[3])
> # channel 1 - no controls
> # channel 2 - geneC and geneD
> posControls[2] <- "(?i)^geneC$|^geneD$"
```

In the constitutive channel R_1, there is one negative control, named *geneA*, and no positive controls. In the pathway-specific reporter R_2 there are two different negative controls (*geneA* and *geneB*), and two diffferent positive controls (*geneC* and *geneD*). Each of the arguments `posControls` and `negControls` should be defined as a vector of regular expressions with the same length as the number of channels in slot `assayData`. These arguments will be passed to the *regexpr* function for pattern matching within the well annotation given in `wellAnno(x)`.

Finally, we construct the quality report pages for the raw data in a directory called `raw`, in the working directory:

```
> out <- writeReport(raw=x, outdir="raw",
+                    posControls=posControls, negControls=negControls)
```

After this function has finished, we can view the index page of the report:

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

3.1 Preprocessing work-flow for two-channel screens

The preprocessing work-flow for two-channel RNAi screens using *cellHTS2* package is shown below:

(a) (optional) Per-plate normalization of each individual channel to remove plate and/or edge effects using function *normalizePlates*.

(b) Channel summarization: the per-plate raw or corrected intensities in each channel are combined using *summarizeChannels*.

(c) (optional) Per-plate normalization of the channel summarized values to remove plate and/or edge effects. This can be done using function *normalizePlates*.

(d) Scoring of replicate measurements (for example, compute z-score values) using the function *scoreReplicates*.

(e) Summarization of replicates (for example, take the median value) using the function *summarizeReplicates*.

The per-plate normalization steps (**(a)** and **(c)**) are optional since they depend on the type of the data and on the channels summarization method to apply (step **(b)**). In particular, step **(a)** is more optional than step **(c)**, since for the simplest channels summarization case, where we take the ratio $\frac{R_2}{R_1}$ (or $log_2\left(\frac{R_2}{R_1}\right)$) between channels intensities, step **(a)** is not required. However, this initial step **(a)** of per-plate correction prior to channels summarization should be performed when we want to apply a more complex summarization function that, for example, makes use of parameters estimated based on overall (i. e. , across plates) intensities. Such case is illustrated for our data set, where we regard a small intensity measurement in the constitutive channel R_1 as a viability defect, excluding it.

For details about the normalization steps performed via the function *normalizePlates* and the available normalization methods, please refer to the main vignette accompanying this package.

In the above preprocessing work-flow, we apply step **(d)** before step **(e)** so that the summary selected for replicate summarization has the same

meaning independently of the type of the assay.

Returning to our data set, in order to distinguish between changes in the readout caused by depletion of specific pathway components versus changes in the overall cell number, we summarize the channels intensities by normalizing the pathway-inducible readout (R_2) against the constitutive reporter (R_1) - step **(b)**. Since in this experiment, reporter 1 (R_1) monitors cell viability, wells with low intensities in R_1 should be masked: these cells are not responding to a specific perturbation of the studied signaling pathway, but show a more unspecific cell viability phenotype. There is no obvious choice for a threshold for the minimum intensity R_1 that we consider still viable; here, we choose to set this cut-off as a low quantile (5%) of the overall distribution of intensity values in R_1 channel. To determine such intensity threshold for R_1 channel, we first need to remove the plate-to-plate variations (step **(a)**) and therefore make the distribution of intensities in the three plates comparable. This is performed by applying plate median scaling to each replicate and channel:

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

Then, we define the intensity cut-off as follows:

```
> ctoff <- quantile(Data(xn)[,,1], probs=0.05, na.rm=TRUE)
```

Figure 1 shows the plate median corrected intensities in R_2 versus R_1 channels, together with the calculated threshold and the positive and negative controls of the pathway-inducible reporter R_2. The wells with intensity values below the calculated threshold are shown in grey and will be set to "NA".

The above procedure of data masking and channel summarization can be carried out at once using the function *summarizeChannels*:

```
> xn1 <- summarizeChannels(xn, fun = function(r1, r2,
+                                thresh=quantile(r1, probs=0.05, na.rm=TRUE))
+                            ifelse(r1>thresh, r2/r1, as.numeric(NA)))
```

The summarized channel intensities are stored in the slot `assayData`. And we can see that the obtained `cellHTS` object contains only one channel:

```
> dim(Data(xn1))
```

```
Features  Samples Channels
    1152        2        1
```

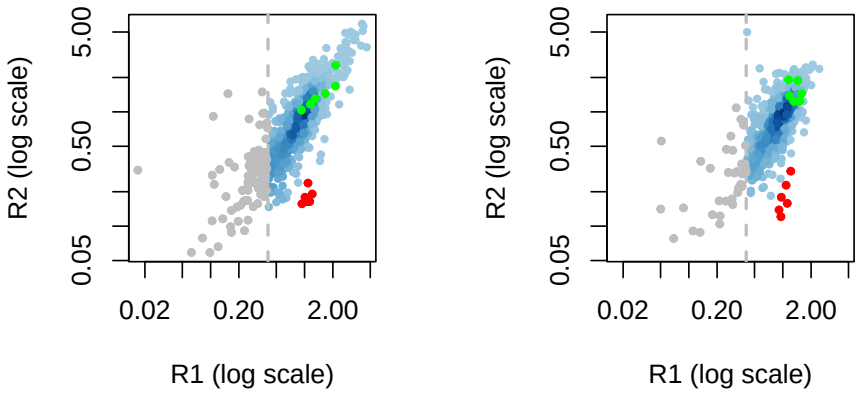

Figure 1: Scatterplot of the plate median corrected intensity values in the signal-dependent channel (R_2) against the plate median corrected intensity values in the constitutive channel (R_1) for replicate 1 (left) and replicate 2 (right). Masked values are shown in grey, while positive and negative controls are shown in red and green, respectively.

After channel summarization, we apply step **(b)**. In this particular case, we take the $\log_2$ and re-apply plate median scaling (in this case, the plate median correction consists of subtracting each plate value by the median of values in that plate, since after log_2 transformation the data are in an additive scale):

```
> xn1 <- normalizePlates(xn1, scale="multiplicative", log=TRUE, method="median",
+                        varianceAdjust="none")
```

Below, we call functions *scoreReplicates* and *summarizeReplicates* to determine the z-score values for each replicate (step **(d)**), and then summarize the replicated z-score values by taking the average (step **(e)**).

```
> xsc <- scoreReplicates(xn1, sign="-", method="zscore")
> xsc <- summarizeReplicates(xsc, summary="mean")
```

The resulting single z-score value per probe are stored in the slot `assayData` of `xsc`. The left side of Figure 2 shows the boxplots of the z-scores for the different types of probes, while the right side of the figure shows the z-scores for the whole screen as an image plot.

```
> par(mfrow=c(1,2))
> ylim <- quantile(Data(xsc), c(0.001, 0.999), na.rm=TRUE)
> boxplot(Data(xsc) ~ wellAnno(xsc), col="lightblue", outline=FALSE, ylim=ylim)
> imageScreen(xsc, zrange=c(-2,4))
```

Now that the data have been preprocessed, scored and summarized between replicates, we call again *writeReport* and use a web browser to view the resulting report. But first, we have to redefine the positive and negative controls for the normalized data stored in `xn1`, because it now corresponds to a single channel. The controls for the normalized data values are the same as those of the raw data channel R_2.

```
> ## Define the controls for the normalized intensities (only one channel):
> # For the single channel, the negative controls are geneA and geneB
> negControls <- "(?i)^geneA$|^geneB$"
> posControls <- "(?i)^geneC$|^geneD$"

> setSettings(list(platelist=list(intensities=list(include=TRUE)),
+                  screenSummary=list(scores=list(range=c(-4,4)))))
> out <- writeReport(raw=x, normalized=xn1, scored=xsc,
+                    outdir="logRatio",
+                    map=TRUE,
+                    posControls=posControls, negControls=negControls)
```

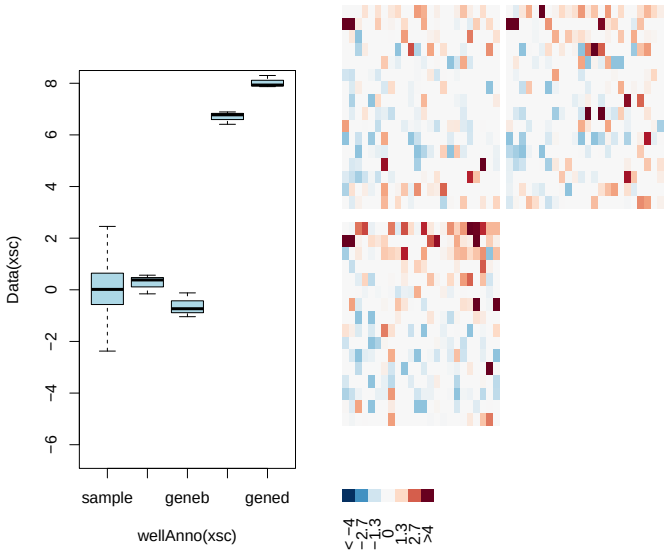

Figure 2: z-scores for the screen. Left Panel: Boxplots of z-scores for the different types of probes. Right Panel: Screen-wide image plot.

The quality reports have been created in the folder `logRatio` in the working directory.

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

The quality reports have been created in the folder `logRatio` in the working directory. Finally, we will save the scored and summarized *cellHTS* object to a file.

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

4 Session info

This document was produced using:

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

- R version 4.2.1 Patched (2022-07-09 r82577), `x86_64-apple-darwin17.0`

- Locale: `C/en_US.UTF-8/en_US.UTF-8/C/en_GB/en_US.UTF-8`
- Running under: `macOS Big Sur ... 10.16`
- Matrix products: default
- BLAS:
  `/Library/Frameworks/R.framework/Versions/4.2/Resources/lib/libRblas.0.dylib`
- LAPACK:
  `/Library/Frameworks/R.framework/Versions/4.2/Resources/lib/libRlapack.dylib`
- Base packages: base, datasets, grDevices, graphics, grid, methods, stats, stats4, utils
- Other packages: AnnotationDbi 1.60.0, Biobase 2.58.0, BiocGenerics 0.44.0, Category 2.64.0, GO.db 3.16.0, IRanges 2.32.0, Matrix 1.5-1, RColorBrewer 1.1-3, S4Vectors 0.36.0, cellHTS2 2.62.0, genefilter 1.80.0, hwriter 1.3.2.1, locfit 1.5-9.6, splots 1.64.0, vsn 3.66.0
- Loaded via a namespace (and not attached): BiocManager 1.30.19, Biostrings 2.66.0, DBI 1.1.3, GSEABase 1.60.0, GenomeInfoDb 1.34.0, GenomeInfoDbData 1.2.9, KEGGREST 1.38.0, KernSmooth 2.23-20, R6 2.5.1, RBGL 1.74.0, RCurl 1.98-1.9, RSQLite 2.2.18, Rcpp 1.0.9, XML 3.99-0.12, XVector 0.38.0, affy 1.76.0, affyio 1.68.0, annotate 1.76.0, assertthat 0.2.1, bit 4.0.4, bit64 4.0.5, bitops 1.0-7, blob 1.2.3, cachem 1.0.6, cli 3.4.1, colorspace 2.0-3, compiler 4.2.1, crayon 1.5.2, digest 0.6.30, dplyr 1.0.10, fansi 1.0.3, farver 2.1.1, fastmap 1.1.0, generics 0.1.3, ggplot2 3.3.6, glue 1.6.2, graph 1.76.0, gtable 0.3.1, hexbin 1.28.2, httr 1.4.4, labeling 0.4.2, lattice 0.20-45, lifecycle 1.0.3, limma 3.54.0, magrittr 2.0.3, memoise 2.0.1, munsell 0.5.0, pillar 1.8.1, pkgconfig 2.0.3, png 0.1-7, preprocessCore 1.60.0, rlang 1.0.6, scales 1.2.1, splines 4.2.1, survival 3.4-0, tibble 3.1.8, tidyselect 1.2.0, tools 4.2.1, utf8 1.2.2, vctrs 0.5.0, xtable 1.8-4, zlibbioc 1.44.0

References

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