qRT-PCR Analysis in R - webinar

Matt Mulé → [mpm@med.unc.edu](mailto:mpm@med.unc.edu); [mulemp@nih.gov](mailto:mulemp@nih.gov);

September 12, 2018

This document is a guide for analyzing RT-PCR results in R. The code in this analysis reflects the plate set up in the powerpoint and the exact code will not run correctly on other analysis than the associated example data. Saving an R script enables you to simply change one line, a file path to the data, to rerun the analysis for every replicate experiment. This becomes important for 384 well / microfluidic based platforms where many genes are profiled simultaneously and calculating fold change values in a speadsheet will be cumbersome. Having a single analysis for every array or plate also ensures reroducability as anyone with your original data can reproduce your results.

First set up R by loading required packages. The As long as this script is located in the same directory as the results directory, it will run as is without changing any file paths etc (the package “here” comes in handy for this - you can learn more about “here” here… <https://www.rdocumentation.org/packages/here/versions/0.1>)

## Loading required package: here

## here() starts at /Users/matthewmule/Dropbox/RTPCR\_webinar/RTPCR\_webinar

## ── Attaching packages ─────────────────────────────────────────────────────────────────────────────────────────── tidyverse 1.2.1 ──

## ✔ tibble 1.4.2 ✔ purrr 0.2.5  
## ✔ tidyr 0.8.1 ✔ dplyr 0.7.6  
## ✔ readr 1.1.1 ✔ stringr 1.3.1  
## ✔ tibble 1.4.2 ✔ forcats 0.3.0

## Warning: package 'dplyr' was built under R version 3.5.1

## ── Conflicts ────────────────────────────────────────────────────────────────────────────────────────────── tidyverse\_conflicts() ──  
## ✖ dplyr::filter() masks stats::filter()  
## ✖ dplyr::lag() masks stats::lag()

Now we load in the data to check out the format for visualization and manipulation. This is similar to the raw output file from any given thermocycler, though there will be a few header lines in the raw results that you can remove manually subset rows of the data frame with something like: data.frame <- dataframe[c(-1:-4), ] #this would get rid of the first 4 rows.

You will also have to add the names of genes, assays, and samples. You can either set this up in the software for the thermocycler or manually add names to a spreadsheet. In this example I assume we looked at the -RT control wells and noted that there was no expression beyond background. These rows are thus eliminated from the example data for simplicity. You can remove these wells with the same type of subsetting in your experiment.

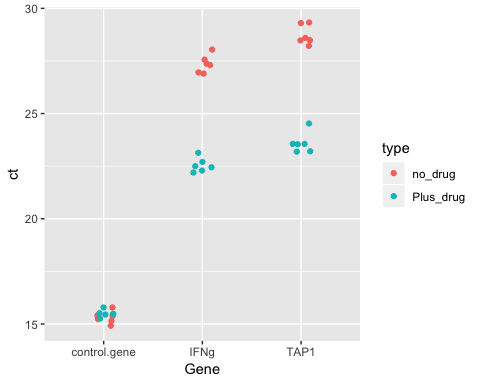
First, We’ll look at the structure of the data. We see that columns or “variables” in R are the well position in the plate, Gene name, sample name, type (the experimental variable) and finally the ct value.

# read in the expression data frame into R from the current directory.   
ge <- read.csv(file = "gene.expressionR.csv", header = T)  
ge

## Well.Position Gene Sample.Name type ct  
## 1 A1 IFNg sample\_1 no\_drug 27.0  
## 2 A2 IFNg sample\_2 no\_drug 27.0  
## 3 A3 TAP1 sample\_1 no\_drug 28.5  
## 4 A4 TAP1 sample\_2 no\_drug 28.5  
## 5 A5 control.gene sample\_1 no\_drug 15.0  
## 6 A6 control.gene sample\_2 no\_drug 15.3  
## 7 B1 IFNg sample\_1 no\_drug 27.4  
## 8 B2 IFNg sample\_2 no\_drug 27.3  
## 9 B3 TAP1 sample\_1 no\_drug 29.4  
## 10 B4 TAP1 sample\_2 no\_drug 28.2  
## 11 B5 control.gene sample\_1 no\_drug 15.1  
## 12 B6 control.gene sample\_2 no\_drug 15.8  
## 13 C1 IFNg sample\_1 no\_drug 28.1  
## 14 C2 IFNg sample\_2 no\_drug 27.5  
## 15 C3 TAP1 sample\_1 no\_drug 29.3  
## 16 C4 TAP1 sample\_2 no\_drug 28.4  
## 17 C5 control.gene sample\_1 no\_drug 15.2  
## 18 C6 control.gene sample\_2 no\_drug 15.3  
## 19 D1 IFNg sample\_3 Plus\_drug 22.4  
## 20 D2 IFNg sample\_4 Plus\_drug 23.1  
## 21 D3 TAP1 sample\_3 Plus\_drug 23.2  
## 22 D4 TAP1 sample\_4 Plus\_drug 23.6  
## 23 D5 control.gene sample\_3 Plus\_drug 15.2  
## 24 D6 control.gene sample\_4 Plus\_drug 15.4  
## 25 E1 IFNg sample\_3 Plus\_drug 22.5  
## 26 E2 IFNg sample\_4 Plus\_drug 22.3  
## 27 E3 TAP1 sample\_3 Plus\_drug 23.2  
## 28 E4 TAP1 sample\_4 Plus\_drug 24.5  
## 29 E5 control.gene sample\_3 Plus\_drug 15.4  
## 30 E6 control.gene sample\_4 Plus\_drug 15.4  
## 31 F1 IFNg sample\_3 Plus\_drug 22.2  
## 32 F2 IFNg sample\_4 Plus\_drug 22.7  
## 33 F3 TAP1 sample\_3 Plus\_drug 23.5  
## 34 F4 TAP1 sample\_4 Plus\_drug 23.5  
## 35 F5 control.gene sample\_3 Plus\_drug 15.8  
## 36 F6 control.gene sample\_4 Plus\_drug 15.6

We want to take a quick peek at the data to get an intuitive feel for the experiment. Does it look like the treatment group had siginficantly different values for our control gene? How are the data distributed? Do replicates have low variation? Clearly we’re going to need to do some sorting and subsetting to answer these questions. Note that conveniently, RT-PCR is generally “tidy” data (see <http://vita.had.co.nz/papers/tidy-data.html>), making it easy to manipulate and plot with the dplyr and ggplot packages which we loaded into our environnment by calling library(tidyverse).

jitter <- position\_jitter(width = 0.1, height = 0.1)  
  
#directly plot the raw data dotplot with factor x variable = genes and label points by the "type" variable.   
ggplot(ge,   
 aes(Gene, ct, color = type)) +  
 geom\_point(position = jitter)



Our replicate pints look to be reasonably close otgether. It appears there is a difference in the treatment vs control groups but this is raw data, we need to normalize these ct values to expression of the control gene.

Next we willl calculate ∆ct values for the treatment vs control groups normalized by expression of our control gene.

Organize the data and take the mean of the technical replicates:

# remember we defined the variable "ge"" above as our raw data frame from the csv file:   
# ge <- read.csv(file = "gene.expressionR.csv", header = T)  
  
ge.mean = ge %>%   
 group\_by(Gene, Sample.Name, type) %>%   
 summarize(avg = mean(ct))  
ge.mean

## # A tibble: 12 x 4  
## # Groups: Gene, Sample.Name [?]  
## Gene Sample.Name type avg  
## <chr> <chr> <chr> <dbl>  
## 1 control.gene sample\_1 no\_drug 15.1  
## 2 control.gene sample\_2 no\_drug 15.5  
## 3 control.gene sample\_3 Plus\_drug 15.5  
## 4 control.gene sample\_4 Plus\_drug 15.5  
## 5 IFNg sample\_1 no\_drug 27.5  
## 6 IFNg sample\_2 no\_drug 27.3  
## 7 IFNg sample\_3 Plus\_drug 22.4  
## 8 IFNg sample\_4 Plus\_drug 22.7  
## 9 TAP1 sample\_1 no\_drug 29.1  
## 10 TAP1 sample\_2 no\_drug 28.4  
## 11 TAP1 sample\_3 Plus\_drug 23.3  
## 12 TAP1 sample\_4 Plus\_drug 23.9

Notice how much easier writing these few lines is than manipulating a spreadsheet.

Now calculate the ∆ct values

dct.ifng <- (ge.mean$avg[5:8] - ge.mean$avg[1:4])  
dct.tap1 <- (ge.mean$avg[9:12] - ge.mean$avg[1:4])  
  
  
dct.df <- cbind(ge.mean$Sample.Name[1:4],  
 ge.mean$type[1:4],  
 dct.ifng,  
 dct.tap1) %>%   
as.data.frame()  
  
dct.df

## V1 V2 dct.ifng dct.tap1  
## 1 sample\_1 no\_drug 12.4 13.9666666666667  
## 2 sample\_2 no\_drug 11.8 12.9  
## 3 sample\_3 Plus\_drug 6.9 7.83333333333333  
## 4 sample\_4 Plus\_drug 7.23333333333333 8.4

Once we have ∆Ct values calaulated for our treatment and control groups, we calculate the fold change of our gene(s) of interest relative to a control group, normalized by expression of the housekeeping gene. this is sometimes refered to the 2^-∆∆ct method for reasons that will become clear below.

In this experiment we compare the average expression seen in our control group to the expression seen in each of the samples that were treated with drug.

Note that this is for illustrative purposes and in reality you would want to have data on many more control samples in order to come up with a average “expect” value for the ∆ct of untreated samples. Also note that you would not do this type of averaging if you were comparing a matched case-control experiment in which you had e.g. a tissue biopsy for a single sample at baseline then treated the subject with a drug, and took another biopsy. In that case you would measure the fold change pairwise in baseline vs treatment for each sample. See ppt slide deck for illustration of this.

I’ve tried to name variables in an intuitive way so you can follow the analysis.

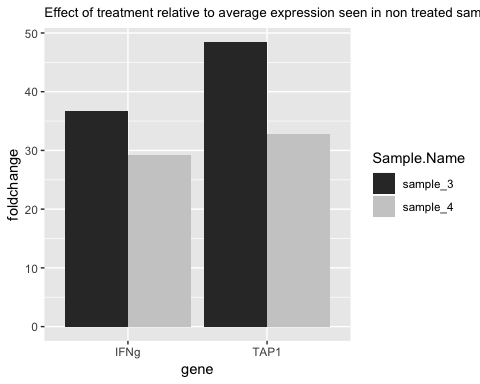
#as noted above, average the ∆ct of the non treated samples   
ifn.av.nodrug <- mean(dct.ifng[1:2])  
tap1.av.nodrug <- mean(dct.tap1[1:2])  
  
# calculate the fold change of the genes of interest relative to the expected value in non treated cells normalized to the control gene   
  
#IFNg  
ddct.ifng <- (as.numeric(dct.df$dct.ifng[3:4]) - ifn.av.nodrug)  
fold.change.ifn <- (2 ^ (-ddct.ifng))  
  
#TAP1  
ddct.tap1 <- (as.numeric(dct.df$dct.tap1[3:4]) - tap1.av.nodrug)  
fold.change.tap1 <- (2 ^ (-ddct.tap1))  
  
# update our results dataframe   
fold\_change <- rbind(fold.change.ifn,fold.change.tap1)%>%   
 t()  
  
dct.df.results = dct.df[3:4, ] %>%   
 cbind(fold\_change)   
  
names(dct.df.results)[1:2] <- names(ge)[3:4]  
  
dct.df.results

## Sample.Name type dct.ifng dct.tap1 fold.change.ifn  
## 3 sample\_3 Plus\_drug 6.9 7.83333333333333 36.75835  
## 4 sample\_4 Plus\_drug 7.23333333333333 8.4 29.17512  
## fold.change.tap1  
## 3 48.50293  
## 4 32.74796

We now see the fold change of the IFN-g and TAP1 genes in the group treated with drug relative to the untreated group normalized to expression of the housekeeping aka control gene.

Now we can plot the results.

results <- gather(dct.df.results, "gene", "foldchange", 5:6)  
results <- results[,-3:-4]  
results$gene <- c("IFNg", "IFNg", "TAP1", "TAP1")  
  
ggplot(data = results) +   
 geom\_col(mapping = aes(x = gene,   
 y=foldchange,   
 fill = Sample.Name),  
 position='dodge') +   
 ggtitle("Effect of treatment relative to average expression seen in non treated samples, normalized by control gene")+  
 theme(plot.title = element\_text(size=10)) + scale\_fill\_grey()



You can now repeat this experiment with independent replicates (meaning different samples, new isolated RNA etc) to measure error.

sessionInfo()

## R version 3.5.0 (2018-04-23)  
## Platform: x86\_64-apple-darwin15.6.0 (64-bit)  
## Running under: macOS High Sierra 10.13.1  
##   
## Matrix products: default  
## BLAS: /Library/Frameworks/R.framework/Versions/3.5/Resources/lib/libRblas.0.dylib  
## LAPACK: /Library/Frameworks/R.framework/Versions/3.5/Resources/lib/libRlapack.dylib  
##   
## locale:  
## [1] en\_US.UTF-8/en\_US.UTF-8/en\_US.UTF-8/C/en\_US.UTF-8/en\_US.UTF-8  
##   
## attached base packages:  
## [1] stats graphics grDevices utils datasets methods base   
##   
## other attached packages:  
## [1] forcats\_0.3.0 stringr\_1.3.1 dplyr\_0.7.6 purrr\_0.2.5   
## [5] readr\_1.1.1 tidyr\_0.8.1 tibble\_1.4.2 tidyverse\_1.2.1  
## [9] ggplot2\_3.0.0 here\_0.1   
##   
## loaded via a namespace (and not attached):  
## [1] Rcpp\_0.12.18 cellranger\_1.1.0 pillar\_1.3.0 compiler\_3.5.0   
## [5] plyr\_1.8.4 bindr\_0.1.1 tools\_3.5.0 digest\_0.6.15   
## [9] lubridate\_1.7.4 jsonlite\_1.5 evaluate\_0.11 gtable\_0.2.0   
## [13] nlme\_3.1-137 lattice\_0.20-35 pkgconfig\_2.0.1 rlang\_0.2.1   
## [17] cli\_1.0.0 rstudioapi\_0.7 yaml\_2.2.0 haven\_1.1.2   
## [21] bindrcpp\_0.2.2 xml2\_1.2.0 httr\_1.3.1 withr\_2.1.2   
## [25] knitr\_1.20 hms\_0.4.2 rprojroot\_1.3-2 grid\_3.5.0   
## [29] tidyselect\_0.2.4 glue\_1.3.0 R6\_2.2.2 fansi\_0.2.3   
## [33] readxl\_1.1.0 rmarkdown\_1.10 modelr\_0.1.2 magrittr\_1.5   
## [37] backports\_1.1.2 scales\_0.5.0 htmltools\_0.3.6 rvest\_0.3.2   
## [41] assertthat\_0.2.0 colorspace\_1.3-2 labeling\_0.3 utf8\_1.1.4   
## [45] stringi\_1.2.4 lazyeval\_0.2.1 munsell\_0.5.0 broom\_0.5.0   
## [49] crayon\_1.3.4