Using My R for fun and profit

Ashton Trey Belew abelew@gmail.com

2014-12-23

Myr vignette!

My first vignette, lets make kittens!

The following block shows how I handle autloading requisite libraries for my code. This makes it easier for me to download/install the R requirements on a new computer, something which I have found myself needing to do more than I would have guessed.

```
## This block serves to load requisite libraries and set some options.
library("myr")
## To set up an initial vignette, use the following line:
## devtools::use_vignette("myr")
autoloads()
```

```
## [1] "Loading biomaRt"
## [1] "Loading BSgenome"
## [1] "Loading BSgenome.Lmajor.friedlin"
## [1] "Loading Cairo"
## [1] "Loading cbcbSEQ"
## [1] "Loading clusterProfiler"
## [1] "Loading data.table"
## [1] "Loading DESeq2"
## [1] "Loading DESeq"
## [1] "Loading devtools"
## [1] "Loading directlabels"
## [1] "Loading DOSE"
## [1] "Loading edgeR"
## [1] "Loading genomeIntervals"
## [1] "Loading ggplot2"
## [1] "Loading GO.db"
## [1] "Loading googleVis"
## [1] "Loading goseq"
## [1] "Loading gplots"
## [1] "Loading gtools"
## [1] "Loading gridExtra"
## [1] "Loading hash"
## [1] "Loading KEGGREST"
## [1] "Loading knitcitations"
## [1] "Loading knitr"
## [1] "Loading knitrBootstrap"
## [1] "Loading methods"
## [1] "Loading motifRG"
## [1] "Loading multtest"
## [1] "Loading pathview"
## [1] "Loading plyr"
## [1] "Loading qvalue"
```

```
## [1] "Loading RamiGO"
## [1] "Loading RColorBrewer"
## [1] "Loading reshape"
## [1] "Loading Rgraphviz"
## [1] "Loading rjson"
## [1] "Loading rmarkdown"
## [1] "Loading robust"
## [1] "Loading roxygen2"
## [1] "Loading Rsamtools"
## [1] "Loading rtracklayer"
## [1] "Loading scales"
## [1] "Loading seqinr"
## [1] "Loading sva"
## [1] "Loading testthat"
## [1] "Loading topGO"
##
## groupGOTerms:    GOBPTerm, GOMFTerm, GOCCTerm environments built.
## [1] "Loading XLConnect"
## [1] "Loading xtable"
```

```
opts_knit$set(progress=TRUE, verbose=TRUE, stop_on_error=FALSE, error=TRUE, fig.width=800/192, fig.heigh
options(java.parameters="-Xmx8g")  ## used for xlconnect -- damn 4g wasn't enough
theme_set(theme_bw(base_size=10))
set.seed(1)
```

Rendering the vignette

The following block has a few lines I use to load data, save it, and render pdf/html reports. I do this under the veritable editor, 'emacs,' with the key combination "Control-c, Control-n" for each line I want to evaluate in R, or "Control-c, Control-c" for a paragraph.

```
load("RData")
rm(list=ls())
save(list=ls(all=TRUE), file="RData")
render("myr.Rmd", output_format="pdf_document")
render("myr.Rmd", output_format="html_document")
```

Tasks that Myr helps me perform

This code was written to speed up and simplify a few specific tasks:

- Reading RNA sequencing count tables (in R/count_tables.R)
- Normalization of data (R/normalization.R)
- Graphing metrics of data to check and evaluate batch effects (R/plots.R)
- Performing contrasts of the data using voom/limma (R/misc_functions.R)
- Plotting RNA abundances by condition/batch (R/plots.R)
- Simplifying ontology/KEGG searches (R/ontology.R)

The following paragraphs will attempt to show how I use it.

Annotation information

Every RNA sequencing experiment I have played with has required a different handling of the genome's annotation. Most, but not all, have kept the data of interest in a gff file. Here is an example of how I process one of those files and make a data frame of genes as well as tooltips, which will be used for googleVis graphs later. In every experiment I have played with, I make a 'reference' directory into which I copy the current annotation data, this way I have a consistent and known version of the annotation. In the example below, this is the TriTrypDB version 8.1 of the T. cruzi genome.

```
tcruzi_annotations = import.gff3("reference/gff/clbrener_8.1_complete.gff.gz")
annotation_info = as.data.frame(tcruzi_annotations)

genes = annotation_info[annotation_info$type=="gene",]
gene_annotations = genes
rownames(genes) = genes$Name
tooltip_data = genes
tooltip_data = tooltip_data[,c(11,12)]
tooltip_data$tooltip = paste(tooltip_data$Name, tooltip_data$description, sep=": ")
tooltip_data$tooltip = gsub("\\+", " ", tooltip_data$tooltip)
rownames(tooltip_data) = tooltip_data$Name
tooltip_data = tooltip_data[-1]
tooltip_data = tooltip_data[-1]
colnames(tooltip_data) = c("name.tooltip")
head(tooltip_data)
```

Reading count tables

In Dr. El-Sayed's lab, there is a very specific naming convention for RNA sequencing experiments. Every sequencing run has an 'HPGL' (host pathogen genomics lab) identifier. All experiments have associated metadata, including the condition in the experiment, the batch, bioanalyzer reports, etc. When I play with data, I keep all this information in a csv file 'samples.csv' and the processed count-tables for the experiment in a specific directory: processed_data/. Therefore, I have a couple functions which automate the import of data into R in the hopes that no mistakes are made.

Here is an example from a recent experiment.

```
samples = read.csv("data/all_samples.csv")
knitr::kable(head(samples))
```

Sample.ID	Type	Stage	Replicate	Media	SRA	Reads	Passed	ncRNA	ncRNA	Remaining	Ge
HPGL0406	WT	EL	1	THY		19026277	353992	1.86%	18672285	17810587	95.39%	366984	1.97%
HPGL0407	WT	EL	2	THY		15074073	259613	1.72%	14814460	14334043	96.76%	269649	1.82%
HPGL0408	mga	EL	1	THY		17112233	293752	1.72%	16818481	15769581	93.76%	318127	1.89%
HPGL0409	mga	EL	2	THY		18298278	339862	1.86%	17958416	16553148	92.17%	349869	1.95%
HPGL0149	WT	LL	1	THY		39107368	8055417	20.60%	31051951	26285560	84.65%	324903	1.05%
HPGL0150	WT	LL	2	THY		35429033	3705275	10.46%	31723758	30012962	94.61%	310449	0.98%

Since I didn't want to copy over all my count tables, you, dear reader, will have to trust that there is a file for each entry in the above table which corresponds to the Sample.ID. These may be organized by sample name or condition. The following code shows how I create an expressionset and fill it with the count data.

```
all_expt = create_expt("all_samples.csv", suffix=".htseq.gz")
## or in another instance
all_human_expt = create_expt("human_samples.csv", suffix="_hg19.count.gz", by_type=TRUE, genes=human_to
```

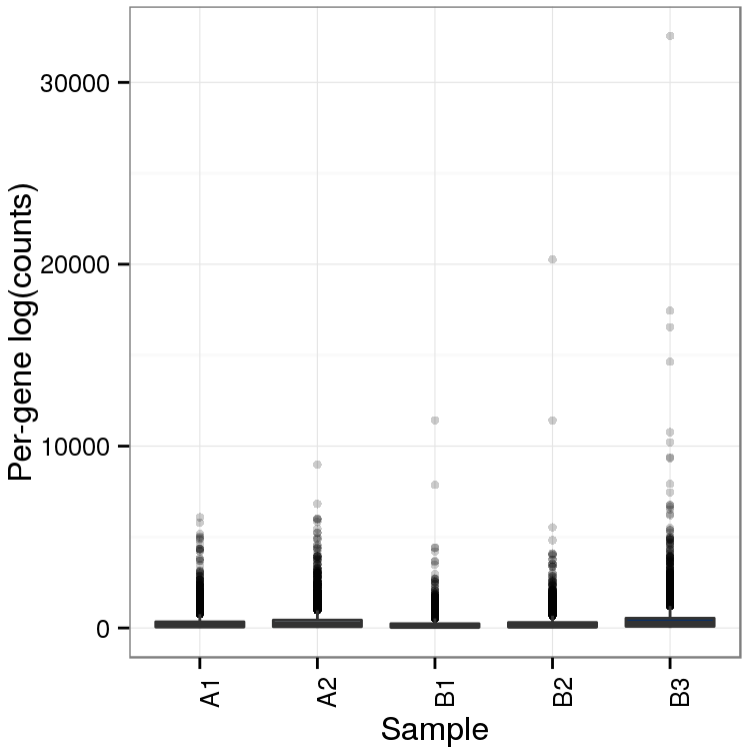
Examining data

Once the data is read in, the first task is always to look at it and evaluate for batch effects and thus decide what to do about them. However, different normalization methods are appropriate in different data sets, therefore I have some functions which attempt to make this easier. For this, I will make a dummy data set using limma's makeExampleData()

```
fun = as.matrix(counts(makeExampleCountDataSet()))
## graph_metrics will perform all these graphs, normalize the data
## re-perform all these graphs, and do an initial batch correction,
## and again plot them.
## Sadly, it requires an expt class as input, so here are just a few
## examples.
my_colors = colorRampPalette(brewer.pal(ncol(fun),"Dark2"))(ncol(fun))
fun_boxplot = my_boxplot(df=fun)
```

```
## Using id as id variables
```

```
print(fun_boxplot)
```

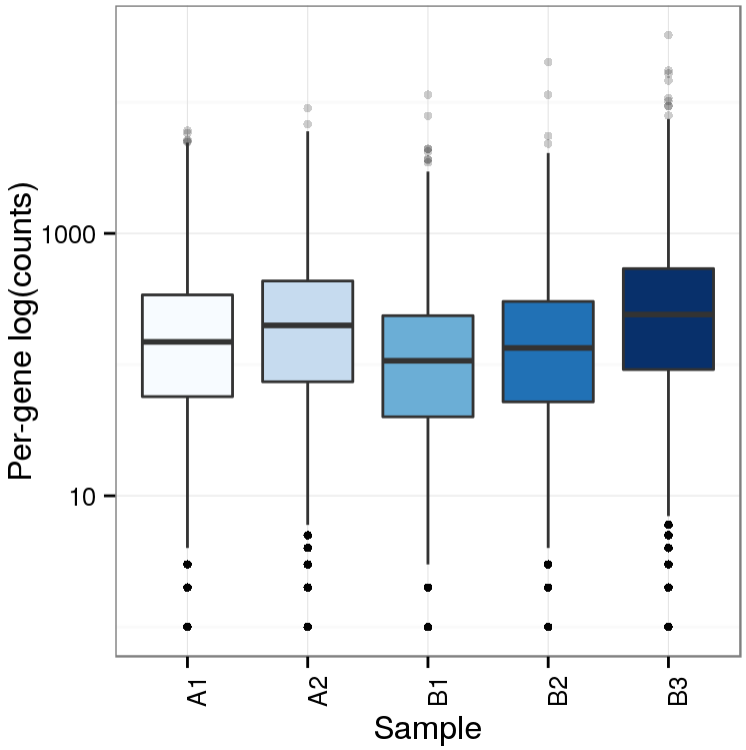



```
log_boxplot = my_boxplot(df=fun, scale="log")
```

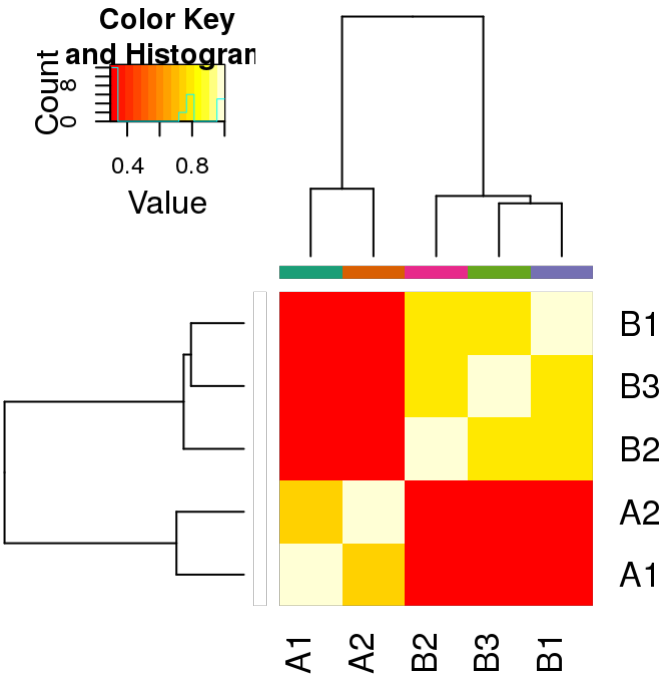
```
## Using id as id variables
```

```
print(log_boxplot)
```

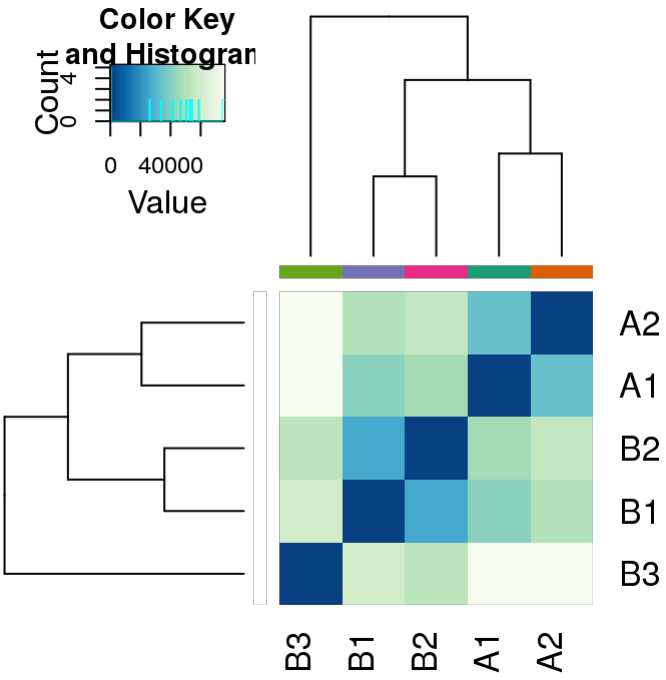
```
## Warning: Removed 267 rows containing non-finite values (stat_boxplot).
```



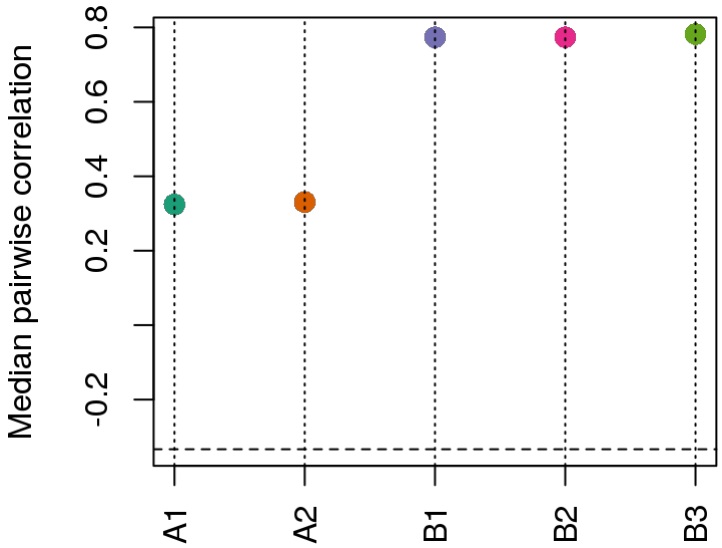

```
## Something is weird in my R session, I changed these functions to not require
## the colors= argument but the R session does not seem to be seeing my change
## despite me running devtools::load_all("~/myr")  oh well
my_corheat(df=fun, colors=my_colors)
```




```
my_disheat(df=fun, colors=my_colors)
```

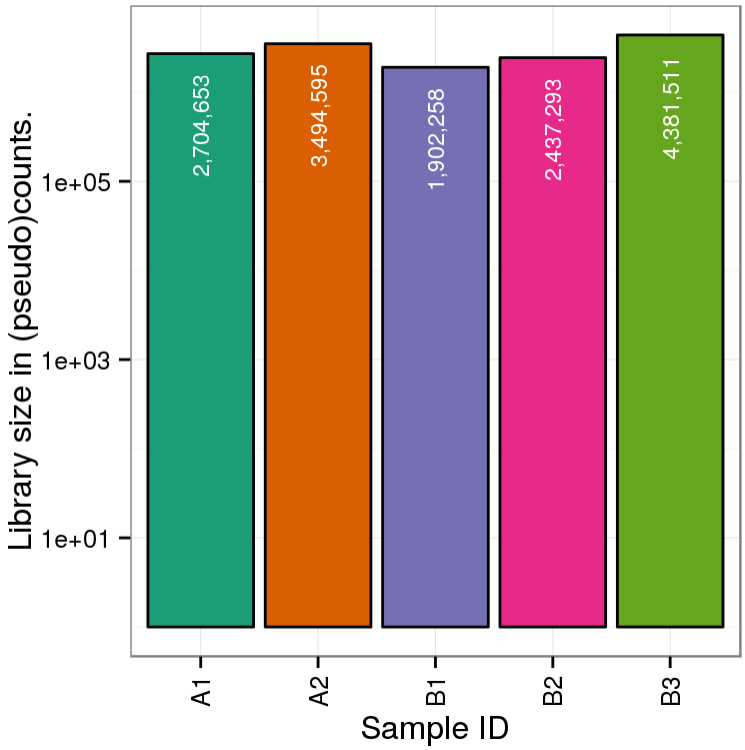



```
my_smc(df=fun, colors=my_colors)
```




```
my_libsize(df=fun)
```

```
## [1] "Adding log10"
```

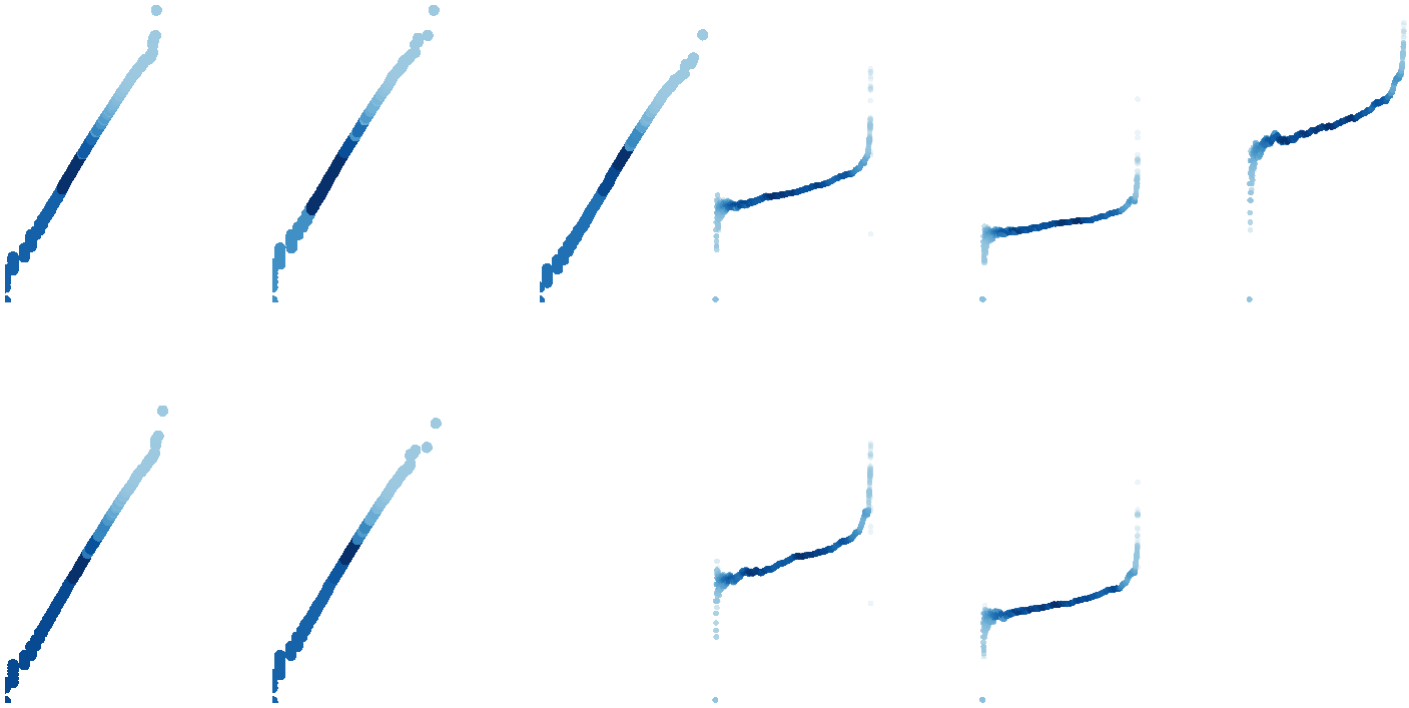



```
my_qq_all(df=fun)
```

```
## Warning in KernSmooth::bkde2D(x, bandwidth = bandwidth, gridsize = nbin, :
## Binning grid too coarse for current (small) bandwidth: consider increasing
## 'gridsize'
```

```
## Warning in KernSmooth::bkde2D(x, bandwidth = bandwidth, gridsize = nbin, :
## Binning grid too coarse for current (small) bandwidth: consider increasing
## 'gridsize'
```

```
## Warning in KernSmooth::bkde2D(x, bandwidth = bandwidth, gridsize = nbin, :
## Binning grid too coarse for current (small) bandwidth: consider increasing
## 'gridsize'
```



```
## $logs
##
## $ratios
##
## $medians
## $medians[[1]]
## [1] -0.2624
##
## $medians[[2]]
## [1] 0.02623
##
## $medians[[3]]
## [1] -0.5953
##
## $medians[[4]]
## [1] -0.37
##
## $medians[[5]]
## [1] 0.2198
```

```
## There are other graphs which require a design or an expt class.
## my_pca, sample_heatmap, my_smd, graph_nonzero
```

Normalizing data

RNAseq data must be normalized. Here is one easy method:

```
## normalize_expt will do this on the expt class, replace the expressionset therein, and
## make a backup of the data inside the expt class.
ql2cpm = my_norm(df=fun)  ## defaults to log2(CPM(quantile(data)))
head(ql2cpm)
```

```
## An object of class "DGEList"
## $counts
##            A1  A2  B1  B2   B3
## gene_1_F   50 341 159 151  301
## gene_2_T  115  96 273 198 1073
## gene_3_F   49  28  27  44   30
## gene_4_F   39  92  16  31   20
## gene_5_F   68  82 100 125  196
## gene_6_F 1629 708 637 747 1098
##
## $samples
##    group lib.size norm.factors
## A1     1  2702976            1
## A2     1  3492339            1
## B1     1  1901003            1
## B2     1  2435775            1
## B3     1  4378926            1
```

```
## size factor, tmm, rle, upperQuartile all require a design matrix.
```

Voom/limma etc

The following are examples of some fun analyses. They generally want a full expressionset and so I won't tell R to evaluate the following block.

```
## expt_subset() will pull pieces out of an expt class for simple comparisons
epi_subset = expt_subset(all_qcpml2, "condition=='clbr_epi'|condition=='cl14_epi'")
## graph the metrics of a subset of the data
epi_graphs = graph_metrics(expt=epi_subset)
epi_graphs$norm_pcaplot
epi_graphs$norm_disheat
epi_graphs$norm_corheat
## Simple comparison will take the first condition as control and the second
## as experimental
epi_comparison = simple_comparison(epi_subset)
epi_table = epi_comparison$table
epi_comparison$pvalue_histogram
epi_comparison$volcano_plot
epi_comparison$ma_plot
epi_comparison$coefficient_scatter
```

A cell-means model using all conditions and batches

```
## acb stands for "kept_conditions_batches"  which takes too long to
## type when setting up the contrasts.
acb = paste0(kept_qcpml2$conditions, kept_qcpml2$batches)
kept_data = exprs(kept_qcpml2$expressionset)
table(acb)
## The invocation of table() keptows me to count up the contribution of
## each condition/batch combination to the whole data set.

## Doing this (as I understand it) means I do nothave to worry about
## balanced samples so much, but must be more careful to understand
## the relative contribution of each sample type to the entire data
## set.

complete_model = model.matrix(~0 + acb)
complete_fit = lmFit(kept_data, complete_model)
complete_voom = my_voom(kept_data, complete_model)
complete_voom$plot
complete_model
## This is an example of what happens when I have heterogenous numbers of samples
## on each side of a contrast, so that a normal design matrix of conditions + batches
## would not work, so instead I add up the contributions of each batch (capital letters)
## and average them out, then use the resulting terms in the various contrasts below.
epi_cl14 = "acbcl14_epiF"
epi_clbr = "acbclbr_epiE"
tryp_cl14 = "(acbcl14_trypB + acbcl14_trypD + acbcl14_trypG) / 3"
tryp_clbr = "acbclbr_trypG"
a60_cl14 =  "(acbcl14_a60A * 2/3) + (acbcl14_a60B * 1/3)"
a60_clbr = "acbclbr_a60A"
a96_cl14 = "acbcl14_a96C"
a96_clbr = "acbclbr_a96C"
epi_cl14clbr = paste0("(",epi_cl14,")", "  -  ", "(",epi_clbr,")")
tryp_cl14clbr = paste0("(",tryp_cl14,")", "  -  ", "(",tryp_clbr,")")
a60_cl14clbr = paste0("(",a60_cl14,")", "  -  ", "(",a60_clbr,")")
a96_cl14clbr = paste0("(",a96_cl14,")", "  -  ", "(",a96_clbr,")")
epitryp_cl14 = paste0("(",tryp_cl14,")", "  -  ", "(",epi_cl14,")")
epitryp_clbr = paste0("(",tryp_clbr,")", "  -  ", "(",epi_clbr,")")
epia60_cl14 = paste0("(",a60_cl14,")", "  -  ", "(",epi_cl14,")")
epia60_clbr = paste0("(",a60_clbr,")", "  -  ", "(",epi_clbr,")")
a60a96_cl14 = paste0("(",a96_cl14,")", "  -  ", "(",a60_cl14,")")
a60a96_clbr = paste0("(",a96_clbr,")", "  -  ", "(",a60_clbr,")")
a60tryp_cl14 = paste0("(",tryp_cl14,")", "  -  ", "(",a60_cl14,")")
a60tryp_clbr = paste0("(",tryp_clbr,")", "  -  ", "(",a60_clbr,")")
## The following contrast is messed up in some as of yet unknown way.
epitryp_cl14clbr = paste0("(",epitryp_cl14,")", "  -  ", "(",epitryp_clbr,")")
## So I will add some more contrasts using data which doesn't get screwed up
epia60_cl14clbr = paste0("(",epia60_cl14,")", "  -  ", "(",epia60_clbr,")")
a60tryp_cl14clbr = paste0("(",a60tryp_cl14,")", "  -  ", "(",a60tryp_clbr,")")
a60a96_cl14clbr = paste0("(",a60a96_cl14,")", "  -  ", "(",a60a96_clbr,")")

complete_contrasts_v2 = makeContrasts(
    epi_cl14=epi_cl14,
```

```
    epi_clbr=epi_clbr,
    tryp_cl14=tryp_cl14,
    tryp_clbr=tryp_clbr,
    a60_cl14=a60_cl14,
    a60_clbr=a60_clbr,
    a96_cl14=a96_cl14,
    a96_clbr=a96_clbr,
    epi_cl14clbr=epi_cl14clbr,
    tryp_cl14clbr=tryp_cl14clbr,
    a60_cl14clbr=a60_cl14clbr,
    a96_cl14clbr=a96_cl14clbr,
    epitryp_cl14=epitryp_cl14,
    epitryp_clbr=epitryp_clbr,
    epia60_cl14=epia60_cl14,
    epia60_clbr=epia60_clbr,
    a60a96_cl14=a60a96_cl14,
    a60a96_clbr=a60a96_clbr,
    a60tryp_cl14=a60tryp_cl14,
    a60tryp_clbr=a60tryp_clbr,
    epitryp_cl14clbr=epitryp_cl14clbr,
    epia60_cl14clbr=epia60_cl14clbr,
    a60tryp_cl14clbr=a60tryp_cl14clbr,
    a60a96_cl14clbr=a60a96_cl14clbr,
    levels=complete_voom$design)
## This colnames() is annoyingly necessary to avoid really obnoxious contrast names.
colnames(complete_contrasts_v2) = c("epi_cl14","epi_clbr","tryp_cl14","tryp_clbr","a60_cl14","a60_clbr"
kept_fits = contrasts.fit(complete_fit, complete_contrasts_v2)
kept_comparisons = eBayes(kept_fits)
```

Clean conditions, batches

On the other hand, I would like to perform arbitrary comparisons among my data even when the batches and conditions look good, so I set up my model/contrast matrices a little strangely even then:

```
all_data = exprs(norm_expt$expressionset)
complete_model = model.matrix(~0 + all_human_expt$conditions + all_human_expt$batches)
## Shorten the column names of the model so I don't have to type so much later...
tmpnames = colnames(complete_model)
tmpnames = gsub("all_human_expt[[:punct:]]","", tmpnames)
tmpnames = gsub("conditions","", tmpnames)
colnames(complete_model) = tmpnames
rm(tmpnames)

complete_voom = my_voom(all_data, complete_model)
complete_voom$plot
complete_fit = lmFit(complete_voom, complete_model)

all_contrasts = makeContrasts(
    ## Start with the simple coefficient groupings for each condition
    none4=none4,
    none24=none24,
    none48=none48,
    none72=none72,
```

```
    bead4=bead4,
    bead24=bead24,
    bead48=bead48,
    bead72=bead72,
    maj4=maj4,
    maj24=maj24,
    maj48=maj48,
    maj72=maj72,
    ama4=ama4,
    ama24=ama24,
    ama48=ama48,
    ama72=ama72,
    ## Now do a few simple comparisons
    ## compare beads to uninfected
    beadnone_4=bead4-none4,
    beadnone_24=bead24-none24,
    beadnone_48=bead48-none48,
    beadnone_72=bead72-none72,
    majnone_4=maj4-none4,
    majnone_24=maj24-none24,
    majnone_48=maj48-none48,
    majnone_72=maj72-none72,
    amanone_4=ama4-none4,
    amanone_24=ama24-none24,
    amanone_48=ama48-none48,
    amanone_72=ama72-none72,
    ## compare samples to beads
    majbead_4=maj4-bead4,
    majbead_24=maj24-bead24,
    majbead_48=maj48-bead48,
    majbead_72=maj72-bead72,
    amabead_4=ama4-bead4,
    amabead_24=ama24-bead24,
    amabead_48=ama48-bead48,
    amabead_72=ama72-bead72,
    ## (x-z)-(a-b)
    ## Use this to compare major and amazonensis
    amamaj_bead_4=(ama4-bead4)-(maj4-bead4),
    amamaj_bead_24=(ama24-bead24)-(maj24-bead24),
    amamaj_bead_48=(ama48-bead48)-(maj48-bead48),
    amamaj_bead_72=(ama72-bead72)-(maj72-bead72),
    ## (c-d)-(e-f) where c/d are: (amazon|major/none)/(beads/none)
    majbead_none_4=(maj4-none4)-(bead4-none4),
    majbead_none_24=(maj24-none24)-(bead24-none24),
    majbead_none_48=(maj48-none48)-(bead48-none48),
    majbead_none_72=(maj72-none72)-(bead72-none72),
    amabead_none_4=(ama4-none4)-(bead4-none4),
    amabead_none_24=(ama24-none24)-(bead24-none24),
    amabead_none_48=(ama48-none48)-(bead48-none48),
    amabead_none_72=(ama72-none72)-(bead72-none72),
    levels=complete_voom$design)
all_fits = contrasts.fit(complete_fit, all_contrasts)
all_comparisons = eBayes(all_fits)
```

```
all_table = topTable(all_comparisons, adjust="fdr", n=nrow(all_data))
write.csv(all_comparisons, file="excel/all_tables.csv")
## write_limma() is a shortcut for writing out all the data structures
all_comparison_tables = write_limma(all_comparisons, excel=FALSE)
```

Ontology searches

The following is an example of a simplified GO search given 20 groups of genes which are from an unannotated organism, but for which blast2GO was performed.

```
for (iter in 1:20) {
    ## Pull the nth group
    subgroup = subset(ontology_info, group == iter)
    subgroup = subgroup[,c("transcript_id","start","end")]
    colnames(subgroup) = c("ID","start","end")
    subgroup_go = simple_goseq(subgroup, lengths=gene_lengths, goids=go_ids)
    ## Save my pictures as images/go_plots-groupN.pdf
    plot_filename = paste("images/go_plots-group", iter, ".pdf", sep="")
    ## I put a block inside the pdf() call just so I can
    ## have indentation to see the set of things being
    ## printed to the pdf file.
    group_trees = goseq_trees(subgroup, subgroup_go)
    pdf(file=plot_filename)
    ## Print the p-value histogram describing how >99% of the go
    ## categories are not in fact significant.
    print(subgroup_go$pvalue_histogram)
    ## mfp_plot/bpp_plot/ccp_plot are bar graphs showing the
    ## 'score' (ratio of genes with in the ontology / all genes) as a bar
    ## and ontology p-value as color.  Thus bigger bars with brighter colors
    ## are considered cooler in the context of the ontology.
    ## Please note that these are explicitly limited to the top ~12 categories
    ## in order to make sure they are readable.
    print(subgroup_go$mfp_plot)
    print(subgroup_go$bpp_plot)
    print(subgroup_go$ccp_plot)
    print(group_trees$MF)
    print(group_trees$BP)
    print(group_trees$CC)
    ## The file/sink operations tell goseq_trees to shut up.  It is a very
    ## chatty tool.  goseq_trees feeds the topGO tool the goseq data and
    ## tells it to plot its ontology trees images which describe the
    ## chosen ontology (molecular function/biological process/cell component)
    ## and the 'interesting' genes therein as boxes and colors.
    ##f = file()
    ##sink(file=f)
    ##close(f)
    dev.off()

    ## Write out the data tables in excel files: excel/group_tables-groupN.xls
    xls_filename = paste("excel/go_tables-group", iter, ".xls", sep="")
    sheetname = paste("mf_interesting-group", iter, sep="")
    write_xls(data=subgroup_go$mf_interesting, sheet=sheetname, file=xls_filename)
```

```
    sheetname = paste("mf_subset-group", iter, sep="")
    write_xls(data=subgroup_go$mf_subset, sheet=sheetname, file=xls_filename)
    sheetname = paste("bp_interesting-group", iter, sep="")
    write_xls(data=subgroup_go$bp_interesting , sheet=sheetname, file=xls_filename)
    sheetname = paste("bp_subset-group", iter, sep="")
    write_xls(data=subgroup_go$bp_subset, sheet=sheetname, file=xls_filename)
    sheetname = paste("cc_interesting-group", iter, sep="")
    write_xls(data=subgroup_go$cc_interesting , sheet=sheetname, file=xls_filename)
    sheetname = paste("cc_subset-group", iter, sep="")
    write_xls(data=subgroup_go$cc_subset, sheet=sheetname, file=xls_filename)
}
```

Vignette Info

Note the various macros within the `vignette` setion of the metadata block above. These are required in order to instruct R how to build the vignette. Note that you should change the `title` field and the `\VignetteIndexEntry` to match the title of your vignette.

Styles

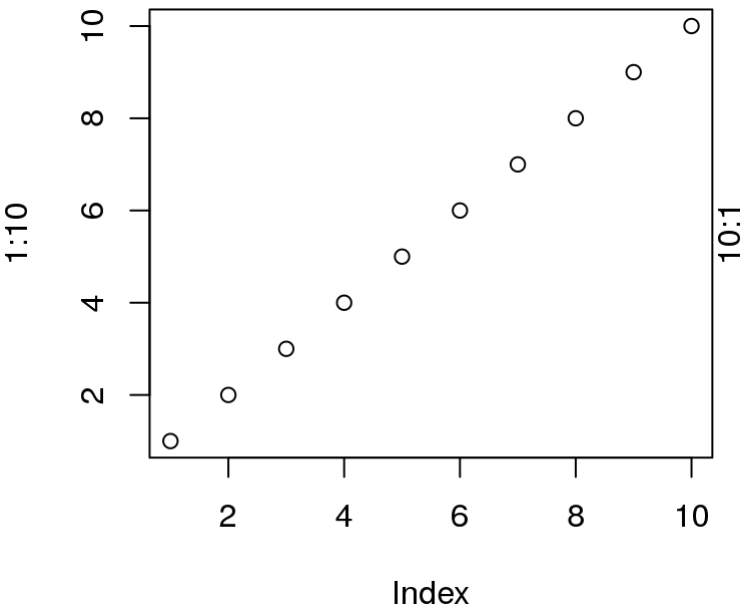
The `html_vignette` template includes a basic CSS theme. To override this theme you can specify your own CSS in the document metadata as follows:

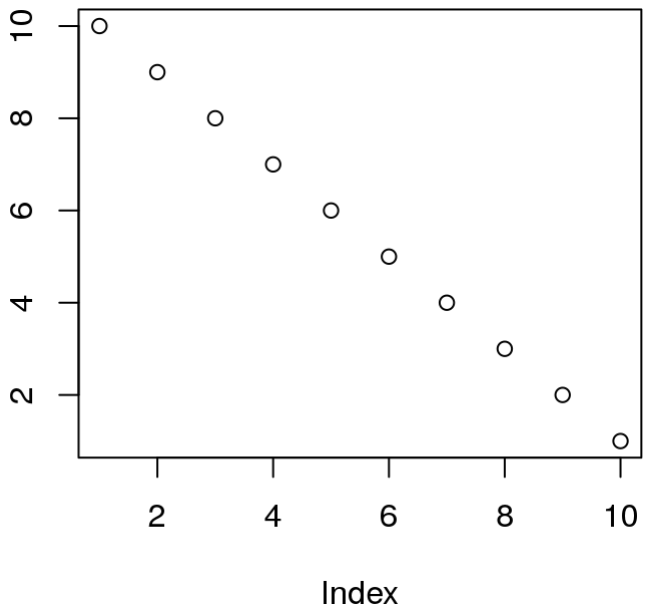
```
output:
  rmarkdown::html_vignette:
    css: mystyles.css
```

Figures

The figure sizes have been customised so that you can easily put two images side-by-side.

```
plot(1:10)
plot(10:1)
```





You can enable figure captions by `fig_caption: yes` in YAML:

```
output:
  rmarkdown::html_vignette:
    fig_caption: yes
```

Then you can use the chunk option `fig.cap = "Your figure caption."` in **knitr**.

More Examples

You can write math expressions, e.g. $Y = X\beta + \epsilon$, footnotes[1], and tables, e.g. using `knitr::kable()`.

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.46	0	1	4	4
Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.02	0	1	4	4
Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1
Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1
Hornet Sportabout	18.7	8	360.0	175	3.15	3.440	17.02	0	0	3	2
Valiant	18.1	6	225.0	105	2.76	3.460	20.22	1	0	3	1
Duster 360	14.3	8	360.0	245	3.21	3.570	15.84	0	0	3	4
Merc 240D	24.4	4	146.7	62	3.69	3.190	20.00	1	0	4	2
Merc 230	22.8	4	140.8	95	3.92	3.150	22.90	1	0	4	2
Merc 280	19.2	6	167.6	123	3.92	3.440	18.30	1	0	4	4

Also a quote using >:

[1] A footnote here.

"He who gives up [code] safety for [code] speed deserves neither." (via)