Approach to barseq of liver and mosquito stage data

Theo Sanderson

```
library(knitr)
opts_chunk$set(tidy.opts=list(width.cutoff=60),tidy=TRUE,warning=F)
```

This, and some code not shown, sets up the starting point in which we have a data frame containing raw barcode counts from all the experiments with data about the pool, condition (=SG,MG, etc.), the mouse and the technical replicate.

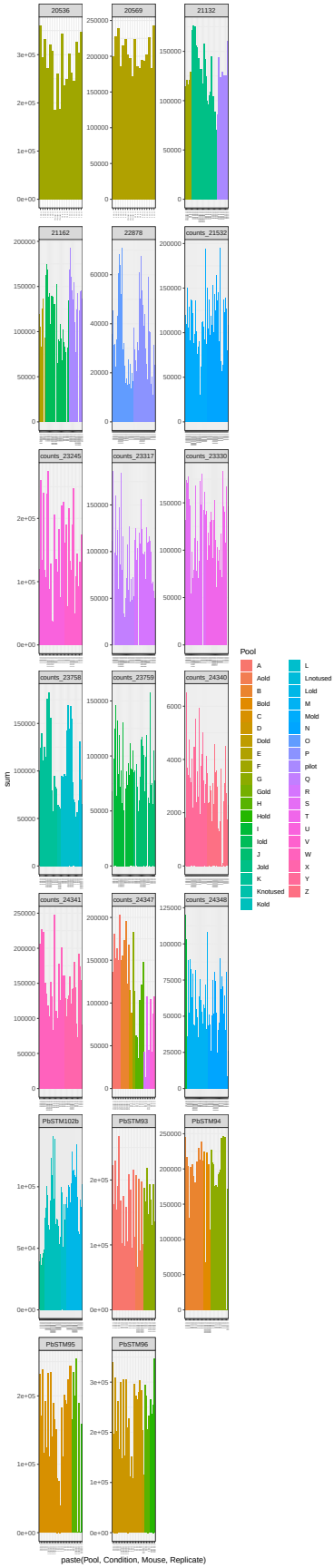
```
# Reshape main_df from wide to narrow format:
narrow <- gather(main_df, key = "gene", value = "value", starts_with("PB"),
  `p230p-tag`, PC_API0037)

narrow$value <- as.numeric(as.character(narrow$value))
narrow$value[is.na(narrow$value)] = 0

inputdata <- narrow %>% filter(narrow$Condition == "Input")
narrow <- narrow %>% filter(Condition %in% c("Passage", "MG",
  "SG", "Rec"))
```

Lets plot how many barcodes we have for each sample to see if everything makes sense:

```
totals <- narrow %>% group_by(Pool, Condition, Mouse, Replicate,
  Run) %>% summarise(sum = sum(value))
ggplot(totals, aes(x = paste(Pool, Condition, Mouse, Replicate),
  fill = Pool, y = sum)) + geom_bar(stat = "identity") + facet_wrap(~Run,
  scales = "free", ncol = 3) + theme_bw() + theme(axis.text.x = element_text(angle = 90,
  hjust = 1, size = 2))
```


20536
20569
21132
21162
22878
counts_21532
counts_23245
counts_23317
counts_23330
counts_23758
counts_23759
counts_24340
counts_24341
counts_24347
counts_24348
PbSTM102b
PbSTM93
PbSTM94
PbSTM95
PbSTM96
sum
paste(Pool, Condition, Mouse, Replicate)
Pool
A
Aold
B
Bold
C
D
Dold
E
F
G
Gold
H
Hold
I
Iold
J
Jold
K
Knotused
Kold
L
Lnotused
Lold
M
Mold
N
O
P
pilot
Q
R
S
T
U
V
W
X
Y
Z

Now we calculate proportions, i.e. the barcode count divided by the total barcode counts for that sample.

```
narrow <- narrow %>% group_by(Pool, Condition, Mouse, Replicate) %>%
  mutate(proportion = value/sum(value))
```

We include only those genes meant to be in the experiment:

```
NewInput<-read.csv("NewInputs.csv")
NewInput<-NewInput %>%
  mutate(Pool = strsplit(as.character(Pool), ",")) %>%
  unnest(Pool)


narrow<-inner_join(narrow,NewInput)
```

```
## Joining, by = c("Pool", "gene")
```

We're going to transform all the proportions into logs to make things easier to work with. But we can't work with zeroes for logs so we add 0.5 to all the values. This is not unreasonable because where we have measured 0 the value could perfectly well actually be 0.5.

```
narrow <- narrow %>% mutate(value = value + 0.5)
```

We calculate proportions again:

```
narrow <- narrow %>% group_by(Pool, Condition, Mouse, Replicate) %>%
  mutate(proportion = (value)/sum(value))
```

And then transform these by taking the log-base-2.

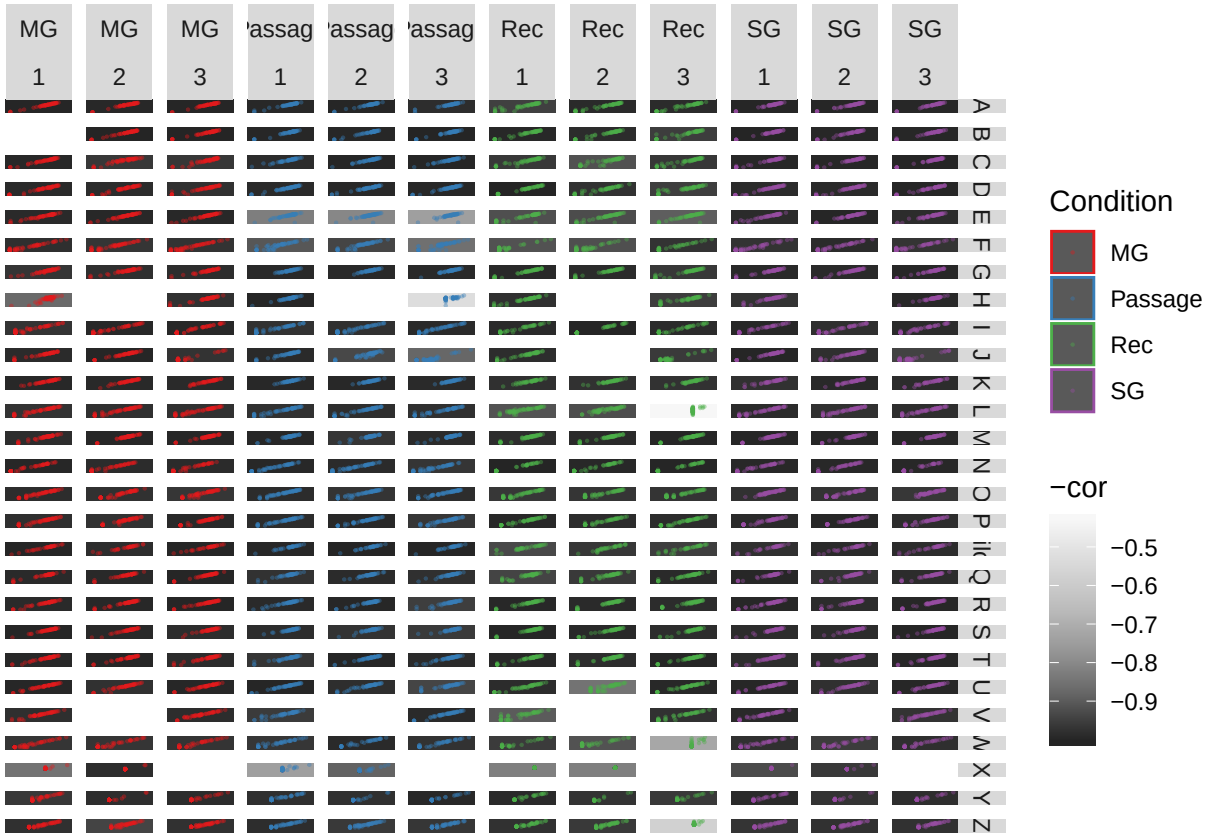
```
narrow<- narrow %>% mutate(log_proportion=log2(proportion))
```

Now we will start to use the replicate information to model variances. We make a new data frame where the two replicates are brought together as 'val1' and 'val2'. We also calculate the mean and standard deviation for each of these pairs. So we have a value for standard deviation for each barcode count, we'll use those later.

```
variance <- narrow %>% group_by(Pool, Condition, Mouse, gene) %>%
  summarise(sd = sd(log_proportion), val1 = log_proportion[1],
    val2 = log_proportion[2], mean_log_proportion = mean(log_proportion))
```

...and work out how well they correlate for each sample to make a plot:

```
cors <- variance %>% group_by(Condition, Mouse, Pool) %>% summarise(cor = cor(val1,
  val2), n = n())
ggplot(variance, aes(x = val1, y = val2, color = Condition, group = Condition)) +
  geom_rect(data = cors, xmin = -50, xmax = 50, ymin = -50,
    ymax = 50, aes(fill = -cor, x = 0, y = 0)) + geom_point(alpha = 0.4,
  size = 0.1) + facet_grid(Pool ~ Condition + Mouse) + scale_fill_distiller(palette = "Greys") +
  theme(panel.background = element_rect(fill = "white")) +
  theme(panel.grid.major = element_blank(), panel.grid.minor = element_blank()) +
  scale_color_brewer(palette = "Set1") + theme(axis.title = element_blank(),
  axis.text = element_blank(), axis.ticks = element_blank())
```




```
ggsave("output/corplot.pdf", width = 10, height = 15)
```

So we know that different samples have different measurement precisions. How do we model the precision with which an individual measurement is made? We have estimates from the standard deviations, calculated from the duplicate PCRs. But these standard deviations are not very accurate measurements of the true precision. They are calculated from only TWO values. Those two values might be very similar by pure chance, indicating high precision wrongly. But we can use the standard deviations from measurements with similar abundances in the same sample to average out this random noise. That is what we do in the next section. We take a "rolling median" across 11 values, after ordering each sample by abundance. Then we enforce monotonicity.

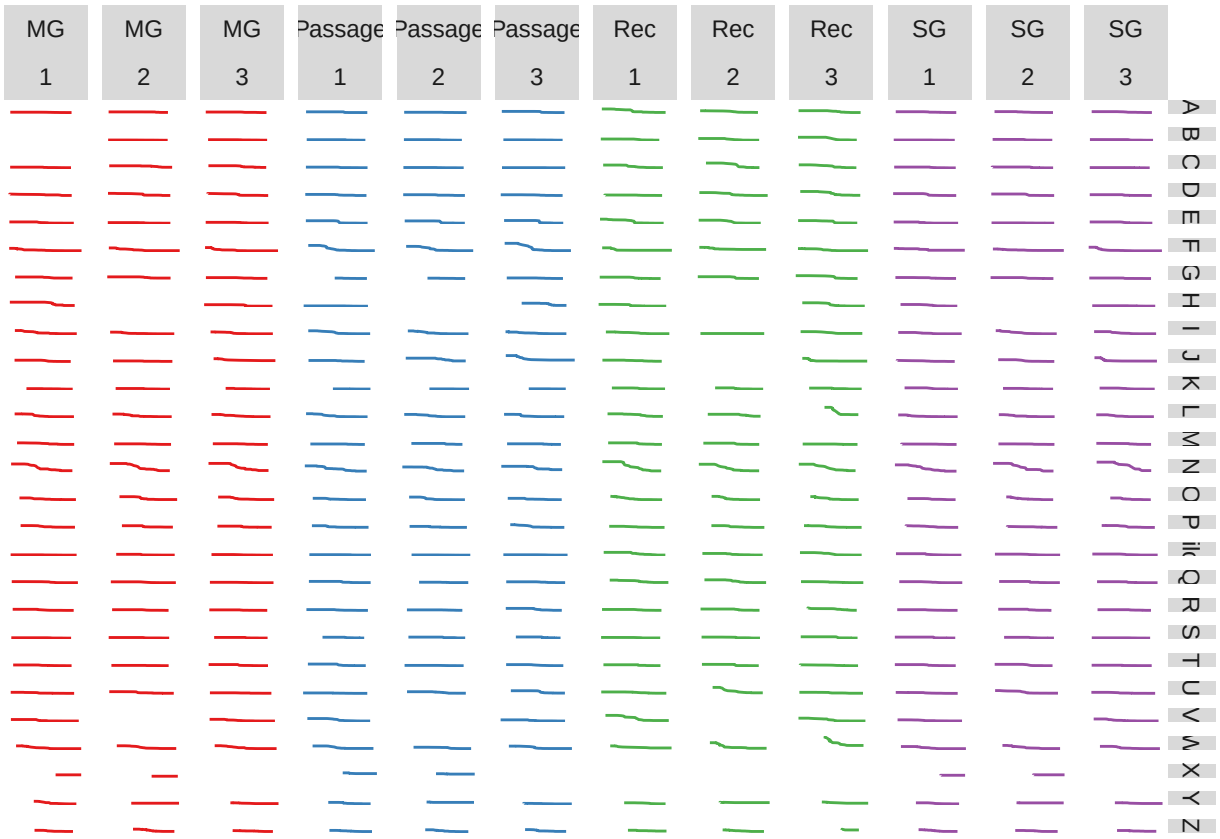
```
variance <- variance %>% group_by(Pool, Condition, Mouse) %>%
  arrange(-mean_log_proportion) %>% mutate(rm_sd = rollmean(x = sd,
  k = 11, fill = "extend"))
variance <- variance %>% mutate(monotonic_rm_sd = cummax(rm_sd))
variance <- inner_join(variance, NewInput)
```

```
## Joining, by = c("Pool", "gene")
```

Let's see what these variance plots look like:

```
ggplot(variance, aes(x = mean_log_proportion, y = monotonic_rm_sd,
  color = Condition)) + geom_line() + facet_grid(Pool ~ Condition +
  Mouse) + theme(panel.background = element_rect(fill = "white")) +
```

```
  theme(panel.grid.major = element_blank(), panel.grid.minor = element_blank()) +
  scale_color_brewer(palette = "Set1") + theme(axis.title = element_blank(),
  axis.text = element_blank(), axis.ticks = element_blank()) +
  guides(color = F)
```



You will notice higher levels for the samples with poor correlations in the previous graph.

Now we want to compare each sample to its previous sample, we define those below.

```
previous = c("Passage", "MG", "SG")
names(previous) <- c("MG", "SG", "Rec")
previous
```

```
##        MG        SG       Rec
## "Passage"      "MG"      "SG"
```

Now we will create a comparison between the next value adn the previous and calculate the difference, and propagate the variance to that measurement of the difference. Because everything is expressed as log-2 a difference of 1 means twice as much and a difference of -1 means half as much.

```
variance$previous = previous[as.character(variance$Condition)]
comparison <- inner_join(ungroup(variance), ungroup(variance),
  by = c("Mouse", "gene", "Pool", Condition = "previous"))

comparison <- comparison %>% mutate(difference = mean_log_proportion.y -
```

```
  mean_log_proportion.x)
comparison <- comparison %>% mutate(differencesd = sqrt(monotonic_rm_sd.y^2 +
  monotonic_rm_sd.x^2))

comparison <- comparison %>% mutate(Condition = Condition.y)
```

At this point we have many measurements of the difference, coming from different mice (and possibly different experiments). We could just take the mean of those values, but it turns out we can do better. We have measurements of the predicted precision with which we have measured each of the differences. We can weight the mean to prioritise the values we think we have measured most accurately. This is the Gaussian mean and the function below defines it. The function calculates both the Gaussian mean and the variance with which we have measured it (it propagates the uncertainty).

```
InverseVarianceWeightedMeanAndVarianceSplit <- function(vals,
  variances, return) {
  df <- data.frame(value = vals, variance = variances)
  df <- df[complete.cases(df), ]

  vals = df$value
  variances = df$variance
  if (length(vals) == 1) {
    var <- variances[1]
    mean <- vals[1]
  } else {

    precs = 1/variances

    mean = sum(vals * precs)/sum(precs)
    # We define two methods of calculating this and choose the larger to be conservative:
    var1 = (1/sum(precs)) * (1/(length(vals) - 1)) * sum((vals -
      mean)^2/variances)
    var2 = 1/sum(precs)
    if (is.na(var1)) {
      var1 <- 0
    }
    var <- max(var1, var2)
  }
  if (return == "mean") {
    return(mean)
  }
  if (return == "variance") {
    return(var)
  }
}
```

We can normalise by assuming the least-reduced genes in each transition represent wildtype.

```
normaliseWithLeastReducedQuarter <- function(unnormaliseddf) {
  comparison <- unnormaliseddf %>% mutate(diffmax = difference +
    2 * differencesd)
  comparison <- comparison %>% mutate(diffmin = difference -
    2 * differencesd)
  norms <- comparison %>% group_by(Pool, Condition, Mouse) %>%
```

```
    arrange(-diffmin) %>% mutate(order = row_number()) %>%
    filter(order < n()/4) %>% summarise(difference = InverseVarianceWeightedMeanAndVarianceSplit(differ
    differencesd^2, "mean"), differencesd = sqrt(InverseVarianceWeightedMeanAndVarianceSplit(difference
    differencesd^2, "variance")))
  norms
  comparison_norm <- inner_join(comparison, norms, suffix = c("",
    "norm"), by = c("Pool", "Condition", "Mouse"))
  comparison_norm <- mutate(comparison_norm, difference = difference -
    differencenorm, differencesd = differencesd + differencesdnorm)

  return(comparison_norm)
}

bloodStageNormalise <- function(df) {
  recipient <- filter(df, Condition == "Rec")
  bloodstage <- read.csv("Barseq20170714.csv")
  recmerge <- inner_join(recipient, bloodstage)
  recmerge <- mutate(recmerge, blood = log2(Relative.Growth.Rate),
    bloodvar = (sqrt(variance)/(Relative.Growth.Rate * log(2)))^2)
  recmerge <- mutate(recmerge, difference = difference - 3 *
    blood, differencesd = sqrt(differencesd^2 + 3^2 * bloodvar))
  recmerge <- recmerge %>% mutate(diffmax = difference + 2 *
    differencesd)
  recmerge <- recmerge %>% mutate(diffmin = difference - 2 *
    differencesd)
  recmerge <- recmerge %>% mutate(power = ifelse(diffmin >
    -1, "notreduced", ifelse(diffmax < (-1), "reduced", "nopower")))
  return(recmerge)
}
```

Instead we'll proceed with the raw-er data: We calculate a confidence interval for the difference using 2xthe standard deviation on either side. We use the confidence intervals to assign phenotypes. If the minimum value is > -1 then the phenotype is not reduced, otherwise it is either: nopower if the max value is > -1 or reduced if the max value is < -1.

```
 mergeMultiples <- function(unmergeddf){
comparisonmerge<- unmergeddf %>% group_by(Pool,Condition,gene) %>% summarise(difference=InverseVariance

comparisonmerge<- comparisonmerge %>% mutate(p=1-pnorm(0,mean=difference,sd=differencesd))
comparisonmerge<- comparisonmerge %>% mutate(diffmax=difference+2*differencesd)
comparisonmerge<- comparisonmerge %>% mutate(diffmin=difference-2*differencesd)
comparisonmerge<- comparisonmerge %>% mutate(power=ifelse(diffmin>-1,"notreduced",ifelse(diffmax<(-1),"
 }
```

We'll export a couple of files to let us look at the data in excel:

```
normalisedQuarterBlood=comparison  %>% filter(Condition=="Rec")%>% normaliseWithLeastReducedQuarter() %
```

```
## Joining, by = "gene"
```

```
normalisedQuarter=comparison  %>% normaliseWithLeastReducedQuarter() %>%  mergeMultiples()
```