Identifying changes in RNA binding using a linear model

In this notebook, we will identify changes in RNA binding following parallel quantification of the "total" and "RNA-bound" protein abundance using the OOPS protocol. We start with a single protein and use the base function `lm`, then apply this to the complete dataset to identify all proteins with a change in RNA binding. Following this, we use the `limma` package to test for RNA binding changes using a moderated t-statistic.

Below we load the required packages and set a plotting theme.

```
# load packages
library(tidyverse)
library(biobroom)
library(MSnbase)
library(limma)

# set up standardised plotting scheme
theme_set(theme_bw(base_size = 20) +
            theme(panel.grid.major=element_blank(),
                  panel.grid.minor=element_blank(),
                  aspect.ratio=1))

cbPalette <- c("#E69F00", "#56B4E9", "#009E73", "#F0E442",
               "#0072B2", "#D55E00", "#CC79A7", "#999999")
```

We start by reading in the data. Our input here is the protein-level quantification for the Nocodazole arrest/release experiment conducted for the OOPS NBT paper (https://www.nature.com/articles/s41587-018-0001-2). In this experiment, we wanted to assess changes in RNA binding in arrested/released cells. To do this, we quantified "total" protein abundance and RNA-bound (extracted by OOPS) protein abundance. The peptide-level abundances have been aggregated to protein level abundance and center-median normalised. Proteins with missing values have been removed. Only proteins quantified in both "total" and "OOPS" samples are included.

The input data here is identical to supplementary table 5 from the above paper.

```
protein_quant_raw <- readxl::read_excel('./ncbi_30607034_OOPS_NBT_table_S5.xlsx',
                                        sheet=3, skip=1, n_max=1917) %>%
  tibble::column_to_rownames('master_protein')
```

We want to store this data in an MSnSet since this is the data structure one would normally use to store proteomics data within R.

As a reminder, the `MSnSet` class mimics the `ExpressionSet` class and contains 3 matrices:

1. assay data (obtained via: `exprs`)
2. feature data (`fData`)
3. phenotype data (`pData`)

The assay data is the quantification of the features (could be PSMs/peptides/proteins) and contains one column per sample

The feature data describes each feature, e.g peptide sequence, master protein accession, retention time etc etc

The phenotype data describes the samples

Below, we read the abundances into the assay data. Then we take the column names of the assay data (the sample names) and separate out the components of these sample names to provide the descriptions for the

samples. Note that we do not have any feature data here since the input data does not decribe any aspects of the features (proteins).

```
protein_quant <- MSnSet(exprs=as.matrix(protein_quant_raw))

pData(protein_quant) <- data.frame(sample_name=colnames(protein_quant_raw)) %>%
  separate(sample_name, into=c('timepoint', 'replicate', 'type'), remove=FALSE) %>%
  tibble::column_to_rownames("sample_name")
```

Let's take a look at our protein quantification data. If we `print` the object, we get a summary including the processing steps performed.

Here we have 1916 features (proteins) quantified across 6 samples.

```
print(protein_quant)
```

```
## MSnSet (storageMode: lockedEnvironment)
## assayData: 1916 features, 18 samples
##   element names: exprs
## protocolData: none
## phenoData
##   sampleNames: 0h_1_total 0h_2_total ... 23h_3_OOPS (18 total)
##   varLabels: timepoint replicate type
##   varMetadata: labelDescription
## featureData: none
## experimentData: use 'experimentData(object)'
## Annotation:
## - - - Processing information - - -
##  MSnbase version: 2.12.0
```

Here's the top of the assay data.

```
print(head(exprs(protein_quant), 2))
```

```
##        0h_1_total 0h_2_total 0h_3_total 6h_1_total 6h_2_total 6h_3_total
## A0AVT1  0.2667887  0.1837573  0.2442799  0.3321695  0.4198107  0.2990757
## A1L0T0 -1.3251187 -0.6965360 -1.1200082 -1.1026241 -1.0478060 -1.3134736
##        23h_1_total 23h_2_total 23h_3_total   0h_1_OOPS   0h_2_OOPS    0h_3_OOPS
## A0AVT1   0.2411216    0.304780   0.2382277 -0.06140054  0.06098304  0.002573485
## A1L0T0  -1.2814785   -1.022792  -1.4183411 -1.15913927 -1.28788959 -1.347853057
##         6h_1_OOPS  6h_2_OOPS  6h_3_OOPS 23h_1_OOPS 23h_2_OOPS 23h_3_OOPS
## A0AVT1  0.8337979  0.7893756  0.6802818  0.6797667  0.5894602  0.6809569
## A1L0T0 -0.8581060 -0.6855888 -1.1143284 -1.3453082 -1.3214604 -1.1930103
```

And here is the phenotype data. As we can see, we have 3 replicates each and three timepoints.

```
print(pData(protein_quant))
```

```
##             timepoint replicate  type
## 0h_1_total         0h         1 total
## 0h_2_total         0h         2 total
## 0h_3_total         0h         3 total
## 6h_1_total         6h         1 total
## 6h_2_total         6h         2 total
## 6h_3_total         6h         3 total
## 23h_1_total       23h         1 total
## 23h_2_total       23h         2 total
## 23h_3_total       23h         3 total
```

```
## 0h_1_OOPS          0h         1  OOPS
## 0h_2_OOPS          0h         2  OOPS
## 0h_3_OOPS          0h         3  OOPS
## 6h_1_OOPS          6h         1  OOPS
## 6h_2_OOPS          6h         2  OOPS
## 6h_3_OOPS          6h         3  OOPS
## 23h_1_OOPS        23h         1  OOPS
## 23h_2_OOPS        23h         2  OOPS
## 23h_3_OOPS        23h         3  OOPS
```

For our purposes, we're only going to focus on the 0h and 6h timepoints phases so we can remove the other data. Below, we subset to the 0h and 6h timepoints

```
protein_quant <- protein_quant[,pData(protein_quant)$timepoint %in% c('0h', '6h')]
```

To run `lm`, we will convert the data format into a "tidy" long format `data.frame` using `biobroom::tidy()`

```
protein_quant_long <- protein_quant %>%
  tidy(addPheno=TRUE)  %>% # "tidy" the object, e.g make it into a tidy data format --> long
  mutate(intensity=value) %>% dplyr::select(-value) # rename the "value" column -> "intensity"
```

Top of the protein expression `data.frame`. See how each intensity value now has it's own row with the other columns describing the associated aspects of the intensity value, e.g the protein and experimental condition

```
print(head(protein_quant_long))
```

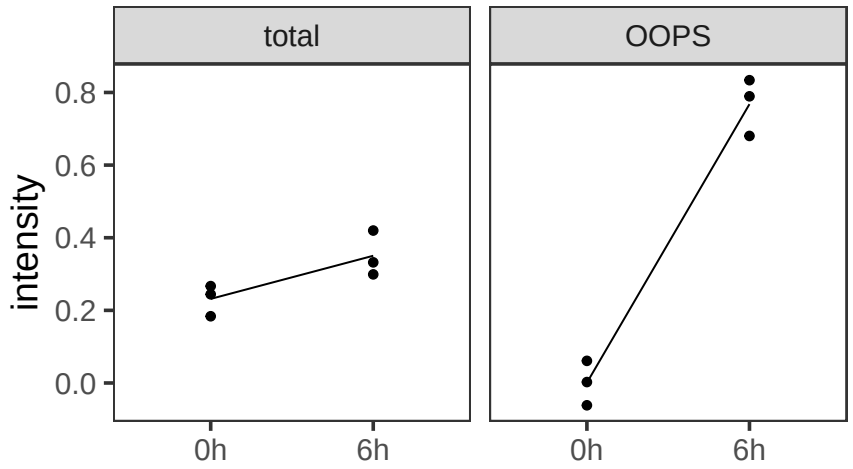
```
## # A tibble: 6 x 6
##   protein sample     timepoint replicate type  intensity
##   <fct>   <fct>      <chr>     <chr>     <chr>     <dbl>
## 1 A0AVT1  0h_1_total 0h        1         total    0.267
## 2 A1L0T0  0h_1_total 0h        1         total   -1.33
## 3 A1L390  0h_1_total 0h        1         total   -0.211
## 4 A1X283  0h_1_total 0h        1         total   -0.114
## 5 A5YKK6  0h_1_total 0h        1         total    0.103
## 6 A6NFI3  0h_1_total 0h        1         total   -0.0490
```

We want to tell R which is the order of the values in the condition and type columns so that the fold changes are in the expected direction, e.g positive = higher in 6h vs 0h.

```
protein_quant_long$timepoint <- factor(protein_quant_long$timepoint, levels=c("0h", "6h"))
protein_quant_long$type <- factor(protein_quant_long$type, levels=c("total", "OOPS"))
```

Now we model the protein intensity according to the models described in `1_simple_example_vd.Rmd`. As an example, let's see the results from just applying the models to a single UniprotID.

```
protein_quant_long %>% filter(protein == 'A0AVT1') %>%
  ggplot(aes(timepoint, intensity, group=1)) +
  geom_point(size=2) +
  stat_summary(geom="line", fun.y=mean) +
  xlab("") +
  facet_wrap(~type)
```




```
fit <- protein_quant_long %>% filter(protein == 'AOAVT1') %>%
  lm(formula=intensity~timepoint*type)

print(summary(fit))
```

```
##
## Call:
## lm(formula = intensity ~ timepoint * type, data = .)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -0.087537 -0.048708  0.007263  0.041451  0.069459
##
## Coefficients:
##                       Estimate Std. Error t value Pr(>|t|)
## (Intercept)            0.23161    0.03620   6.398 0.000209 ***
## timepoint6h            0.11874    0.05119   2.320 0.048951 *
## typeOOPS              -0.23089    0.05119  -4.510 0.001975 **
## timepoint6h:typeOOPS   0.64836    0.07240   8.956 1.92e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.0627 on 8 degrees of freedom
## Multiple R-squared:  0.9673, Adjusted R-squared:  0.955
## F-statistic: 78.86 on 3 and 8 DF,  p-value: 2.78e-06
```

We can see that the model fits the data well ("Multiple R-squared: 0.9673, Adjusted R-squared: 0.955"). We can see that the interaction term that we're interested in for changes in RNA binding ("timepoint6h:typeOOPS") significantly deviates from zero.

Below, we make a function to run the linear model for a single protein

```
Change_in_RNA_binding_LM <- function(obj){

  fit <- obj %>% lm(formula=intensity ~ timepoint + type + timepoint*type)
```

```
  fit_values <- c(coef(summary(fit))["timepoint6h:typeOOPS",],
                  summary(fit)$adj.r.squared)

  names(fit_values) <- c("lm_fold_change", "lm_std_error", "lm_t_value",
                         "lm_p_value", "lm_adj_R_squared")
  fit_values <- as.data.frame(t(fit_values), stringsAsFactors=FALSE)

  return(fit_values)
}
```

Testing the above function with our example protein

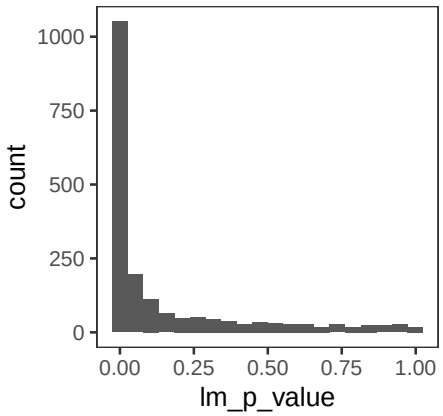
```
protein_quant_long %>% filter(protein == 'AOAVT1') %>% Change_in_RNA_binding_LM()
```

```
##   lm_fold_change lm_std_error lm_t_value   lm_p_value lm_adj_R_squared
## 1      0.6483564   0.07239593   8.955702 1.921547e-05        0.9550228
```

Below, we make a function to run the `Change_in_RNA_binding_LM ()` function on all proteins in turn using dplyr. We will use the standard Benjamini-Hochberg method to adjust p-values for the multiple tests we have conducted.

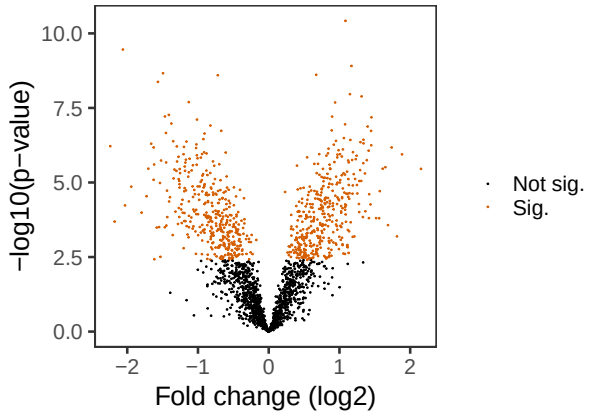
Below, we plot the p-values. Under the null hypothesis they should show an approximately uniform distribution. If there were a large number of proteins with a significant change in RNA binding, we would expect an additional "spike" with low p-values (<0.05). We see an approximately uniform distribution with a large 'spike' of low p-values but also a slight skew towards low p-value. This may indicate the presence of changes in RNA binding which we are insufficiently powered to detect.

```
lm_results %>% ggplot(aes(lm_p_value)) + geom_histogram(bins=20)
```



We can use a volcano plot to take a look at the estimated fold changes and associated p-values

```
lm_results %>%
  mutate(sig=ifelse(lm_BH<0.01, "Sig.", "Not sig.")) %>% # add "sig" column
  ggplot(aes(x=lm_fold_change, y=-log10(lm_p_value), colour=sig)) +
  geom_point(size=0.25) +
  scale_colour_manual(values=c("black", cbPalette[6]), name="") + # manually adjust colours
  xlab("Fold change (log2)") + ylab("-log10(p-value)") # manual axes labels
```



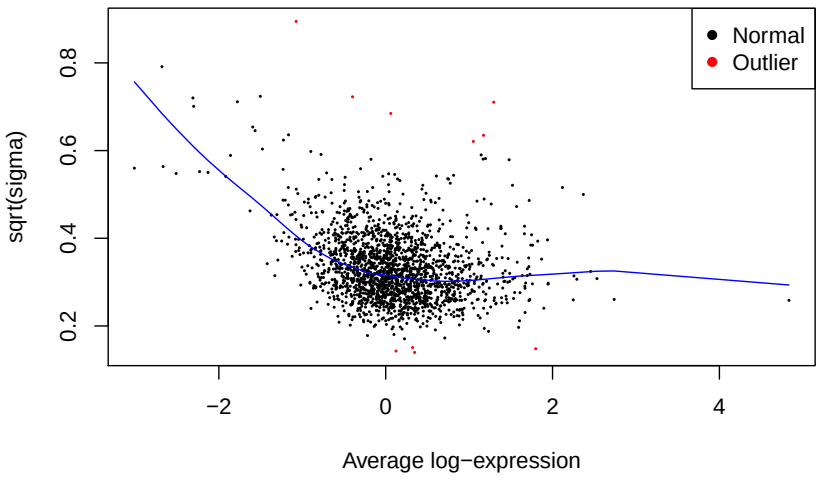
Now, let's apply `limma` to performed a moderated version of the above linear modeling. For a full explanation of the steps in a `limma` analysis, see: https://bioconductor.org/packages/release/bioc/vignettes/limma/inst/doc/usersguide.pdf

First of all we need to create a design matrix. We can do this from the `pData` since this contains the information about the samples

```
timepoint <- factor(protein_quant$timepoint, levels=c("0h", "6h"))
type <- factor(protein_quant$type, levels=c("total", "OOPS"))

design <- model.matrix(~timepoint*type)

limma_fit_results <- lmFit(protein_quant, design)

limma_fit_results <- contrasts.fit(limma_fit_results, coefficients="timepoint6h:typeOOPS")

limma_fit_results <- eBayes(limma_fit_results, trend=TRUE, robust=TRUE) # Bayesian shrinkage
```

Visualising the relationship between protein abundance and variability

```
plotSA(limma_fit_results)
```



Below, we summarise the number of signficant p-values (post BH FDR correction) using a 1% FDR threshold.

```
summary(decideTests(limma_fit_results, p.value=0.01, adjust.method="BH"))
```

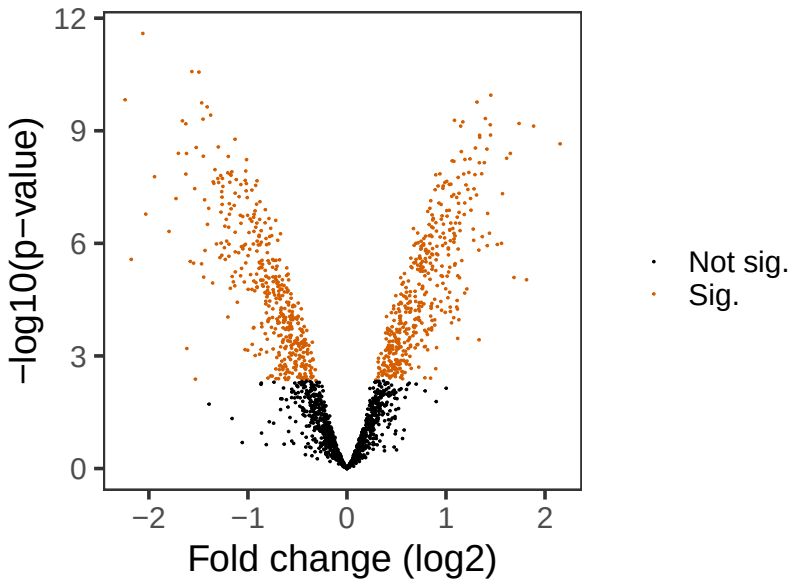
```
##        timepoint6h:typeOOPS
## Down                    439
## NotSig                 1036
## Up                      441
```

Below, we extract the limma results

```
limma_results <- topTable(limma_fit_results, coef = "timepoint6h:typeOOPS", n = Inf, confint=TRUE)
```

And then make the volcano plot for the `limma` RNA binding changes

```
limma_results %>%
  mutate(sig=ifelse(adj.P.Val<0.01, "Sig.", "Not sig.")) %>% # add "sig" column
  ggplot(aes(x=logFC, y=-log10(P.Value), colour=sig)) +
  geom_point(size=0.25) +
  scale_colour_manual(values=c("black", cbPalette[6]), name="") + # manually adjust colours
  xlab("Fold change (log2)") + ylab("-log10(p-value)") # manual axes labels
```



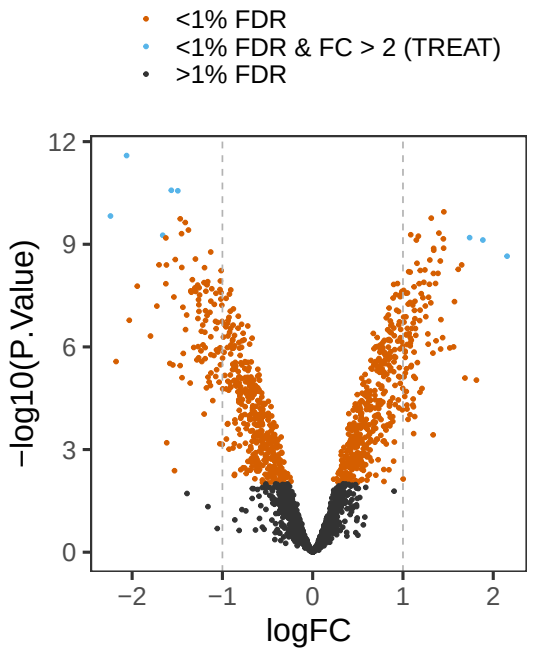
Finally, we can get `limma` to return the signficant changes using both p-value and log-fold change thresholds using the TREAT method to explictly test the null hypothesis that the fold change is less than our specified threshold. (https://www.ncbi.nlm.nih.gov/pubmed/19176553).

```
# Test null hypothesis than change is <2-fold
limma_results_treat <- treat(limma_fit_results,lfc=1)

# subset to proteins passing the 1% FDR threshold for absolute fold change > 2
limma_results_treat_sig <- topTreat(limma_results_treat, coef = "timepoint6h:typeOOPS", n = Inf,
                                    p.value=0.01, lfc=1, adjust.method="fdr", confint=0.95)
```

And below we reproduce our volcano plot including the 95% confidence interval and highlight those proteins which have $< 1\%$ FDR and an absolute fold change significantly greater than 2.

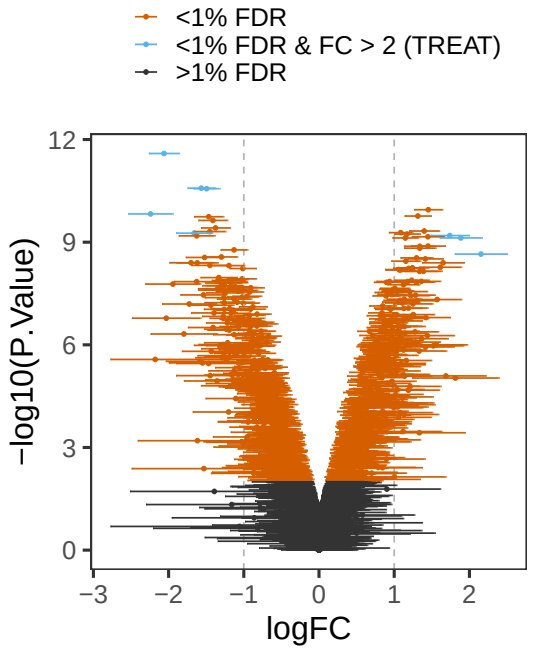
```
limma_results$sig <- ifelse(limma_results$P.Value<=0.01, "<1% FDR", ">1% FDR") # add "sig" column
limma_results$sig[
```

```
  rownames(limma_results) %in% rownames(limma_results_treat_sig)] <- "<1% FDR & FC > 2 (TREAT)"

limma_results$SE <- sqrt(limma_fit_results$s2.post) * limma_fit_results$stdev.unscaled[,1]

p <- limma_results %>%
  ggplot(aes(logFC, -log10(P.Value), colour=sig)) +
  geom_point(size=1) +
  scale_colour_manual(values=c(cbPalette[c(6,2)], "grey20"), name="") + # manually adjust colours
  geom_vline(xintercept=1, linetype=2, colour="grey70") +
  geom_vline(xintercept=-1, linetype=2, colour="grey70") +
  theme(legend.position="top", legend.direction=2)

print(p)
```



We can also add error bars to the above Below, we can see that many of the proteins with a fold change (FC) point estimate > 2 have a 95% confidence interval that overlaps the dashed lines for >2-fold change. TREAT also takes the multiple testing into account so it's even more conservative than just using the 95% CI shown below.

```
print(p + geom_errorbarh(aes(xmin=CI.L, xmax=CI.R)))
```



Of course, the threshold for the fold changes you are interested in depends entirely on your prior expectations. A 2-fold threshold for "biologically" relevant changes will likely be overly conservative in the great majority of cases.