# Lab 3: RNA Sequencing Data

**Grading**: Turn in your first attempt at the tasks for a binary "fair attempt or not" grade on Canvas. That is, your first attempt need not be neat or correct.

**Done early?** Come discuss your work with me and I'll give you one suggestion on how to improve what you've got in the remaining time. Iterate if necessary.

## Scientific context

Consider the following RNA sequencing experiment conducted by your collaborator:

- *Data:* A count matrix with one row for each (biological) sample, one column for each gene, and $(i, j)$th entry indicating the number of sequencing reads that are attributed to gene $j$ in sample $i$; and a condition label for each sample (A or B).
- *Analysis:* For each gene $j$, test for (multiplicative) differential expression by fitting a Poisson GLM with condition as the covariate, and testing the null hypothesis that the condition coefficient is equal to 0.

(Note that the data model and the analysis today are heavily simplified and are not exactly what you will see out in the wild!)

Your collaborator wants to call any gene that has p-value smaller than 0.1 differentially expressed. Their ultimate concern is the proportion of follow-up validation efforts from their RNA-seq experiment that is ultimately wasted. To help them better understand the consequences of this strategy, you decide to design a simulation study to investigate the distribution of the false discovery proportion.

Functions to generate and analyze a single RNA-seq data set on 100 samples (50 from each condition) and 100 genes are provided at the end of this document. You are free to modify them if you see fit.

## Task 1

Decide on a single set of values for the arguments of `generate_RNAseq_data()`; think of this like fixing a particular RNA-seq experiment to repeat. Simulate differential expression results from 200 repetitions of this experiment, and summarize the FDP distribution across the 200 repetitions.

## Task 2

Simulate to investigate how (i) the proportion of non-differentially expressed genes and (ii) the correlation between genes affect the FDP distribution. Summarize your results with a plot.

## Code

## Data generation

Arguments:

- `prop_null`: the proportion of non-differentially expressed genes; for e.g. `prop_null = 90`, there will be $0.9 * 100 = 90$ differentially expressed genes.
- `de_effect`: degree of differential expression; for e.g. `de_effect = 2`, the differentially expressed genes have a mean expression level twice as big in condition B compared to condition A
- `rho`: amount of dependence; for e.g. `rho = 0.5`, the correlation between two genes in each condition is 0.5.

I'm labelling this as "model and data" because I'm separately saving out some information about the generating model that we would not know in real life (which genes are differentially expressed).

In practice, I usually set up my simulations with the pipeline `generate_model() %>% generate_data_from_model() %>% analyze_data() %>% evaluate_results()`. (In part because I'm used to using the simulator package, and in part because I just like the flexibility of designing each component of the infrastructure of a simulation study separately.)

```
library(dplyr)
library(stringr)

generate_model_and_data <- function(prop_null, de_effect, rho) {
    n <- 50
    p <- 100

    num_null <- ceiling(prop_null*p)

    cond <- factor(c(rep("A", n), rep("B", n)))

    Sigma <- (1-rho)*diag(p) + rho*matrix(1, p, p)

    mvn_copula_gen <- matrix(rnorm(2*n*p), 2*n, p, byrow=TRUE)%*%chol(Sigma) %>%
                                        as_tibble(.name_repair = \(x) str_c("gene", 1
                                        mutate(across(everything(), \(x) pnorm(x)))

    pois_data_gen <- mvn_copula_gen %>%
            mutate(across(everything(),
                                function(x) {
                                        ifelse(cond == "B" &
                                                    as.numeric(str_remove(cur_column(),
                                                    qpois(x, lambda=5*de_effect),
                                                    qpois(x, lambda=5))
                                }))) %>%
            mutate(cond=cond) %>%
            select(cond, everything())

  gene <- c(rep("H0", num_null), rep("H1", p - num_null))

  return(list(data=pois_data_gen,gene=gene))
```

```
}

set.seed(2)
ex_model_and_data <- generate_model_and_data(prop_null=0.9, de_effect=2, rho=0)

ex_model_and_data$data
```

```
# A tibble: 100 × 101
   cond  gene1 gene2 gene3 gene4 gene5 gene6 gene7 gene8 gene9 gene10 gene11
   <fct> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>  <dbl>  <dbl>
 1 A         3     5     9     3     5     5     6     4    10      5      6
 2 A         7     5     4     3     3    10     7    10     4      4      3
 3 A         6     3    12     5     3     6     5     4     6      5      7
 4 A         4     4     7     1     7     7     5     7     8      3      6
 5 A         4     0     3     3     7     4     3     1     8      5      5
 6 A         4     6     3     4     5     8     3     8     3      8      2
 7 A         3     5     3     3     8     7     5     9     7      3      4
 8 A         8     5     4     9     5     8     5     5     3      5      1
 9 A         3     1     6     4     5     3     7    11     8      4      4
10 A         5     7     5     5     5     6     6     5     3      4     10
# ℹ 90 more rows
# ℹ 89 more variables: gene12 <dbl>, gene13 <dbl>, gene14 <dbl>, gene15 <dbl>,
#   gene16 <dbl>, gene17 <dbl>, gene18 <dbl>, gene19 <dbl>, gene20 <dbl>,
#   gene21 <dbl>, gene22 <dbl>, gene23 <dbl>, gene24 <dbl>, gene25 <dbl>,
#   gene26 <dbl>, gene27 <dbl>, gene28 <dbl>, gene29 <dbl>, gene30 <dbl>,
#   gene31 <dbl>, gene32 <dbl>, gene33 <dbl>, gene34 <dbl>, gene35 <dbl>,
#   gene36 <dbl>, gene37 <dbl>, gene38 <dbl>, gene39 <dbl>, gene40 <dbl>, …
```

```
table(ex_model_and_data$gene)
```

```
H0 H1
90 10
```

## Data analysis

The following function takes in a RNA-sequencing data set `data` of the format produced by `generate_model_and_data()` and returns a p-value for each gene by fitting a Poisson GLM with condition as the covariate, and testing the null hypothesis that the condition coefficient is equal to 0, and a rejection status for each gene based on thresholding the p-value at `alpha`.

```
analyze_RNAseq_data <- function(data, alpha) {
        data %>%
                summarize(across(-cond, function(gene) broom::tidy(glm(gene ~ cond),
                                family="poisson")[["p.value"]][2])) %>%
                                    tidyr::pivot_longer(cols = everything(), names_to = "
                                                                        values_to = "
                                    mutate(rejects = (pval <= alpha))
```

```
}

analyze_RNAseq_data(ex_model_and_data$data, 0.1)
```

```
# A tibble: 100 × 3
   gene     pval rejects
   <chr>   <dbl> <lgl>
 1 gene1  0.293  FALSE
 2 gene2  0.963  FALSE
 3 gene3  0.0462 TRUE
 4 gene4  0.0646 TRUE
 5 gene5  0.314  FALSE
 6 gene6  0.245  FALSE
 7 gene7  0.239  FALSE
 8 gene8  0.541  FALSE
 9 gene9  1.00   FALSE
10 gene10 0.917  FALSE
# ℹ 90 more rows
```