Neuro-Genomics
Exercise 2 - Sequencing Data Analysis

The purpose of this exercise is to implement some of the algorithms used in sequencing data analysis. We will first implement two algorithms for normalization of sequencing data (Part 1). Then we will implement permutations analysis in the context of detecting circadian genes with defined false positive rate (Part 2). Finally, we will implement hierarchical clustering (Part 3).

<u>General instructions to all sections:</u>

a) For each part in this exercise a code is expected. Run your code on the provided data and make sure that it is running properly.
b) The code for this exercise can be written either in R, Matlab or Python.
c) Make sure that your code is properly explained with comments inside the code. We need to be able to read the code and understand it.
d) Do not use built-in functions for the functions needed below, the idea is to write your own implementations.
e) Submit two functions for Part 1, one script for Part 2, and one function for Part 3. Also submit the output generated by the functions and script, using the input described.

Part 1 - Normalization of sequencing data

Since more than one sample is usually examined in any experiment, the different expression vectors need to be normalized. The easiest normalization method is to account for the fact that different samples can produce different total number of sequencing reads. This simple normalization was covered in exercise 1. In this part you will implement two methods for normalization, which are considered better than the simple total counts normalization.

The first normalization method you will implement is TMM normalization. Specifically, write a function that implements these steps:

1) The function should get as an input two expression vectors, and also a constant D, which is a number between 0 to 1. The first expression vector will be the reference vector and will be termed X, with values Xi. The second vector, which will be normalized according to the reference vector, will be termed Y, with values Yi. You can assume that the order of the vectors (i.e. the rows which correspond to genes) is the same in both vectors.
2) Compute the M folds as follows:
   $Mi = log\frac{Yi}{Xi}$
   No need to compute M fold value in cases where Xi or Yi is equal to zero.
   You can use the natural logarithm (ln).
3) The different M folds are weighted according to the expected deviation in the folds Yi/Xi. Assume Poisson distribution and compute the weights as follows:
   $Wi = sqrt(\frac{Xi+Yi}{2})$
   No need to compute weight value in cases where Xi or Yi equal to zero.
4) Sort the M folds according to their values, and trim the lower D/2 fraction and higher D/2 fraction of folds. For example, assume D is 0.1, in this case only M folds values that are between 0.05 to 0.95 percentile should be considered in the next step.
5) For the trimmed M folds, compute the TMM as follows:
   $TMM = \frac{sum(Wi\times Mi)}{sum(Wi)}$
6) The second vector, i.e. vector Y, should be corrected by dividing all its values by this factor:
   $factor = exp(TMM)$
7) The function should return: i) X, as is; ii) the corrected vector Y'; and iii) the factor of correction.
8) The function should also plot the distribution of values in the two vectors before and after the normalization. You should generate two plots - one for the distribution of values (i.e. histogram) before the normalization. This 'before' plot should show both vectors on the same plot, each one with a different color (that should be explained using legends). The title should be 'vectors histogram before normalization'. The second plot should have the title 'vectors histogram after normalization', and should show both vectors after

normalization on the same plot, each one with a different color (again explained using legends).

9) Run the function on two vectors from the ‘pasilla’ data. Reminder about this data is provided at the end of this part.

The second normalization method you will implement is quantile normalization. Specifically, write a function that implements these steps:

1) The function should get as an input two expression vectors. The first expression vector will be the reference vector and will be termed X, with values Xi. The second vector, which will be normalized according to the reference vector, will be termed Y, with values Yi. You can assume that the order of the vectors (i.e. the rows which correspond to genes) is the same in both vectors.
2) Calculate quantile normalization as described here. For simplicity, you can assume that each expression vector contains only unique values, i.e. the expression vector does not have two values which are the same.
3) The function should return: i) the corrected vector X’; ii) the corrected vector Y’.
4) The function should also plot the distribution of values in the two vectors before and after the normalization. You should generate two plots - one for the distribution of values (i.e. histogram) before the normalization. This ‘before’ plot should show both vectors on the same plot, each one with a different color (that should be explained using legends). The title should be ‘vectors histogram before normalization’. The second plot should have the title ‘vectors histogram after normalization’, and should show both vectors after normalization on the same plot, each one with a different color (again explained using legends).
5) Run the function on two vectors from the ‘pasilla’ data. Choose the same two vectors you used for the TMM normalization. Reminder about this data is provided at the end of this part.

Reminder about the 'pasilla' data

In exercise 1 we used a dataset from an experiment performed on Drosophila melanogaster cell cultures. The experiment was designed to investigate the effect of RNAi knock-down of the splicing factor pasilla (see Conservation of an RNA regulatory map between Drosophila and mammals). Several samples were collected from untreated fly cells (i.e. control samples), and several samples were collected from fly cells that were treated with RNAi. The dataset is stored in the package 'pasilla'.
If you are using R, the ‘pasilla’ data can be obtained below. If you are not using R, a file with the ‘pasilla’ data can be found here.

```
## Install the package 'pasilla'
# copy and paste the lines below into R (if prompted - choose ‘Update all’)
if (!requireNamespace("BiocManager", quietly = TRUE))
  install.packages("BiocManager")
BiocManager::install("pasilla")
## Use the installed library
```

```
# copy and paste the line below into R
library("pasilla")
## Load the actual data from the pasilla package
# copy and paste the lines below into R
pasCts <- system.file("extdata",
                      "pasilla_gene_counts.tsv",
                      package="pasilla", mustWork=TRUE)
pasAnno <- system.file("extdata",
                       "pasilla_sample_annotation.csv",
                       package="pasilla", mustWork=TRUE)
cts <- as.matrix(read.csv(pasCts,sep="\t",row.names="gene_id"))
coldata <- read.csv(pasAnno, row.names=1)
coldata <- coldata[,c("condition","type")]
rownames(coldata) <- sub("fb", "", rownames(coldata))
cts <- cts[, rownames(coldata)]
```

Part 2 - Detect rhythmical patterns using Fourier transform

This part will be a follow up for the analysis you performed in exercise 1, part 5. There you used a fast fourier transform to detect genes that have circadian expression using a time-course dataset.
Specifically, in exercise 1 you used the file CircadianRNAseq.csv which contains the processed RNA sequencing data. In this file each row represents one gene. The first column 'RefSeqID' gives the official gene annotation, and the last column 'GeneSymbol' gives the common name of the gene. Columns 2 to 13 contain the time-course measurements which were performed over two consecutive days ('A' means day 1 and 'B' is day 2). Each value in columns 2 to 13 represents the number of sequencing reads that were aligned against the gene in a given row. However, these values are not raw count data - these are normalized count values (therefore - no further normalization is needed). In exercise 1 you used this file to process all the genes in the dataset and to sort them according to the normalized FFT power in the circadian frequency of 1/24. Let's term the normalized FFT power in the circadian frequency of 1/24 as the "G factor". This is a number between 0 to 1. Here we want to decide which G factor to use as a cutoff in order to a specific true-positive rate.

Write a script that implements these steps:

1) Run the scripts that you already generated for the calculation of the G factor (in exercise 1) on the real dataset. Create a vector that holds the number of genes detected for each G factor cutoff from 0 to 0.99 in steps of 0.01. For example, in this vector the first value will be the number of genes that have a G factor of 0 and above (this will give all the genes). The second value in this vector will be the number of genes that have a G factor of 0.01 and above, and so on. The last value in this vector will be the number of genes that have a G factor of 0.99 and above (this will probably give zero genes).
2) Generate permuted dataset instead of the read dataset. The permuted data should preserve the statistical properties of the dataset but should remove the biological signal, i.e. no real circadian genes should be detected. [What do you think is the best way to permute the data? There are two options: either to shuffle the data between the rows (genes) or to shuffle the data between the columns (the time points). One of these options makes more sense…]
3) Run the scripts that you already generated for the calculation of the G factor (in exercise 1) on the permuted dataset. Create a vector that holds the number of genes detected for each G factor cutoff from 0 to 0.99 in steps of 0.01. For example, in this vector the first value will be the number of genes that have a G factor of 0 and above (this will give all the genes). The second value in this vector will be the number of genes that have a G factor of 0.01 and above, and so on. The last value in this vector will be the number of genes that have a G factor of 0.99 and above (this will probably give zero genes).
4) Repeat steps (2)&(3) 100 times overall. That is, create 100 realizations of the dataset and for each one calculate the vector of the number of genes detected as a function of the G factor. Average the 100 resulting vectors (the vectors from step 3) -- this will give

one averaged vector for the number of genes detected for each G factor, when using non-biological data.

5) Plot the vector from step 1 and the vector from step 4 in the same figure. The Y-axis should be the vector values and the X-axis should be the G factor used as a cutoff (0 to 0.99 with steps of 0.01). Use different colors and legends that explain the colors. Give labels to the axes.
6) For each G factor used as a cutoff (0 to 0.99 with steps of 0.01), calculate the fraction of true-positives. This is the number of genes detected using real data minus the number of genes detected using shuffled data, divided by the number of genes detected using real data. For example, say that 100 genes are detected using real data with G factor 0.7, and 10 genes are detected using shuffled data with the same G factor as a cutoff. Then the true-positives rate will be (100-10)/100=0.9.
7) Plot the fraction of true-positives as a function of the G factor used as a cutoff (0 to 0.99 with steps of 0.01). Give labels to the axes.

Part 3 - Hierarchical clustering

In this part you will implement hierarchical clustering, and use it on the genes detected as circadian in the previous part.

Write a function that implements hierarchical clustering as described below. The input for this function should be: 1) a matrix where each column is one experimental condition and each row is one gene; 2) a list with the names of genes, in the same order as in the matrix; 3) an integer K for the desired number of clusters. Print an error and exist if K>=N, where N is the number of genes.
The output of this function should be the names of the genes in each cluster, when K clusters are obtained. Print the names of the genes in each one of the K clusters in a clear way.
Test this function on all the genes detected as circadian with true-positives rate of 0.8 (at Part 2), with K=6.

<u>Preparation step:</u>
The input is a matrix with the expression profiles of N genes. The expression profile is the expression values across the experimental conditions for that given gene. Calculate the distant matrix d (an N by N matrix). The distant matrix should be calculated using Pearson's correlation between the expression profiles. You are allowed to use a built-in function for the calculation of Pearson's correlation. If the Pearson's correlation between the expression profiles of gene i and gene j is marked R, then the distance $d_{i,j}$ will be 1-R. When the two profiles are the most similar, R=1 and the distance d is zero. When the two profiles are not similar, R=0 and the distance d is 1. When the two profiles are anti-correlated, R=-1 and the distance d is 2.

<u>Iteration:</u>
At the beginning, there are N 'clusters' (each gene is a cluster). Using the procedure below, in each iteration we combine two gene profiles (or two clusters) to form a new cluster.
1. Find gene/cluster i and gene/cluster j that have the smallest distance between them. At the beginning, this is just the smallest element in d.
At later steps (i.e. after we start the clustering), use the matrix D instead of d, which is:

$$D_{i,j} = \frac{1}{n_i + n_j} \sum_{p \in C_i} \sum_{q \in C_j} d(p,q)$$

n_i and n_j are the number of genes in cluster i and cluster j, respectively. The double summation just means that we are running over all members of cluster i (say 2 genes) and all members of cluster j (say 3 genes), and sum all the distances d between them. We then divide by the total number of genes (in our simple example this is 5). Note that when we start the clustering D=d.

2. Connect gene/cluster i and gene/cluster j to a new cluster (ij).
At the beginning the new cluster will have 2 members.

At later steps the number of members will be n(ij) = $n_i + n_j$.
n_i and n_j are the number of genes in cluster i and cluster j, respectively.
Compute the distance from the new cluster to all other genes/clusters (except for i and j, which are no longer relevant) as an average of the distances from its components.
At the beginning, this is just:
D(ij),k = ½*$d_{i,k}$ + ½*$d_{j,k}$
At later steps, in which the number of members in i and j are not necessarily 1, use the weighted average to calculate this distance:

$$D_{(ij),k} = (\frac{n_i}{n_i + n_j})D_{i,k} + (\frac{n_j}{n_i + n_j})D_{j,k}$$

3. Delete the columns and rows in D that correspond to clusters i and j, and add a column and row for cluster (ij), with D(ij),k computed as above.

4. Return to step 1 until only K clusters remain.