Exercises 1-3
Statistics in Genetics

Alfons Edbom Devall

February 7, 2022

Statistics in Genetics, 7.5hp

Contents

1 Chapter 1

1.1 Exercise 1

1.1.1 Background

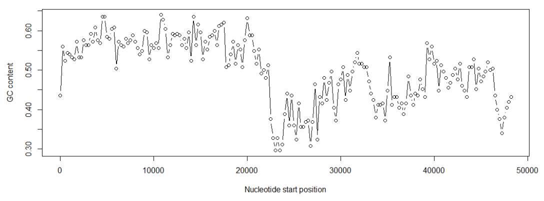
A common first step when analyzing the genomic content of an organisms DNA is to check the percentage of the G- and C bases in the sequence of nucleotides, henceforth called the GC-content. If a region of the sequence has a higher GC-content than the background GC-content, it is likely that a gene/genes can be found there. Since calculating the GC-content is relatively fast and easy, with the potential to give good hints of interesting regions of the genome, this is often the first step when analyzing a new genome.

In this exercise the complete genomic sequence of the Enterobacteria phage lambda (NC_001417) was analyzed for its GC-content with various choices of window sizes.

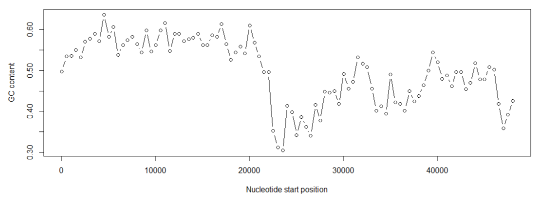
1.1.2 What was done and discussion

In this exercise, the complete genome of Enterobacteria phage lamda was downloaded from GenBank. The genome was then analyzed by using the programming language R. First the FASTA-file was loaded using the seqinr package. Then a function to create a sliding window plot was made, called ”slidingwindowplot”, which takes as input both the window size to use and what sequence the plot should be made of. The effect of using different sizes of the window size then investigated.
To view the code for this exercise, see Appendix A.A

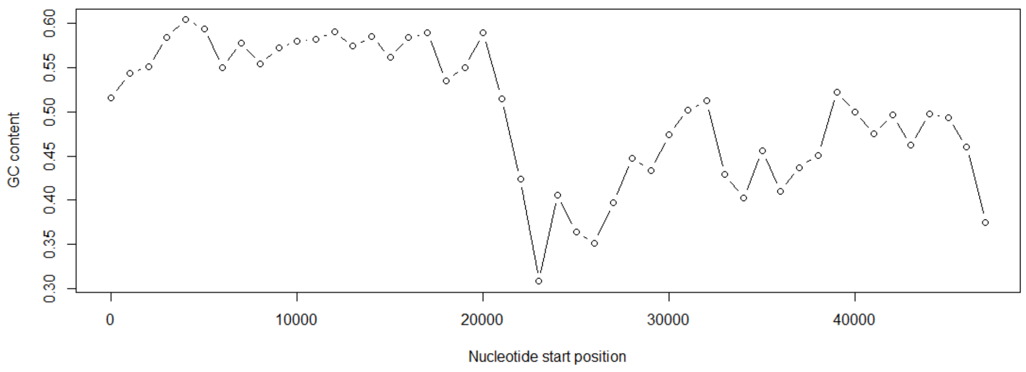
In figure 1 it can be seen that different sizes of the sliding window does appear to change which regions of high GC-content that are found in the genome. With a small size of the sliding window as in 1a, most regions with slightly higher GC-content than the background will be captured, but will instead be a bit cluttered and if they window size gets too small it starts to get too noisy. If using a larger size of the sliding window, as in figure 1c the data is getting more smooth, and as a consequence some of the smaller regions of high GC-content will be missed. To balance out these opposing factors, a window size somewhere in-between can be used, which then will give a good and high resolution figure, that is not too noisy and not too smooth, which can be seen in figure 1b.



(a) 250 bases/window



(b) 500 bases/window



(c) 1000 bases/window

Figure 1 – Sliding window plots of the genome of Enterobacteria phage lamda using different sizes of the sliding window.

2 Chapter 2

2.1 Exercise 1

2.1.1 Background

When studying the genome of an organism, it is often of interest to identify regions of the DNA that codes for a gene.
To identify genes, a good start is to find the open reading frames (ORF) in the subject-DNA. An ORF consists of a DNA-sequence that starts with a start codon (e.g. ATG) and ends with a stop codon (TAA, TAG, TGA), where both the start and stop codon are in the same reading frame. The length of the ORF is then calculated by counting the number of nucleotides from the start codon to the stop codon, for it to be an ORF it needs to be divisible by 3 (since a codon consists of 3 nucleotides).

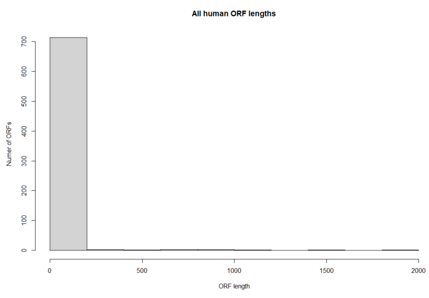
In this exercise the human(NC_001807), chimp(NC_001643) and mouse (NC_005089) mitochondrial DNA (mtDNA) has been investigated. The genetic code for mitochondria in vertebrates is slightly different than the standard one. The differences that are of interest for this exercise is that it has different start and stop codons: In mtDNA the start codons are: ATG and ATA, and the stop codons are: TAA, TAG, AGA and AGG. All ORFs in the mtDNA have been identified and candidate protein coding genes has been selected. Finally, it was calculated how large fraction of the whole mtDNA is covered by the candidate genes.

2.1.2 What was done and discussion

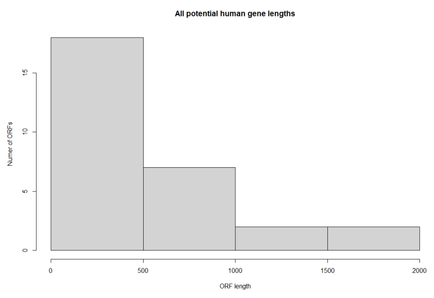
In this exercise the complete mtDNA from human, chimp and mouse was downloaded from GenBank. The mtDNA was then analyzed by using the programming language R. First the ORFik package was used to identify all ORFs in the sequence. A histogram of the lengths of the found ORFs was then done and can be seen in the figures 2a, 3a and 4a. Then the minimum length of the found ORFs was increased in small increments until I felt it was an appropriate number of ORFs left and also had an good approximation for how short a gene could potentially be. I chose 40 codons (120 base pairs) to be the minimum length to classify a found ORF as a candidate protein coding gene, and a histogram of the ORF lengths can be seen in the figures 2b, 3b and 4b.

When all potential protein coding genes had been identified, the fraction of the whole mtDNA sequence that was covered by the genes was calculated. This was done by summing up the lengths of all genes and then dividing by the whole sequence length. The resulting portions for human, chimp and mouse mtDNA was: 86.5%, 89.1% and 86.7% respectively.

To solve this exercise a few different approaches could have been taken that may have given a more accurate result. When choosing what ORFs to be considered as genes, it might have been a better approach to generate random sequences of similar length to the mtDNA sequences and then seen how many ORFs at different lengths one could expect by chance and then choose a minimum length depending on that result.
When calculating what portion of the mtDNA that is covered by potential protein coding genes one could also take into account which regions of the actual mtDNA sequence is covered by each gene. This would most likely give a significantly more accurate reading of how large portion of the mtDNA sequence codes for genes. This is because the sequence is read in both directions(forwards and backwards) and thus contains some overlapping regions that right now is calculated twice.
To view the code for this exercise, see Appendix A.B

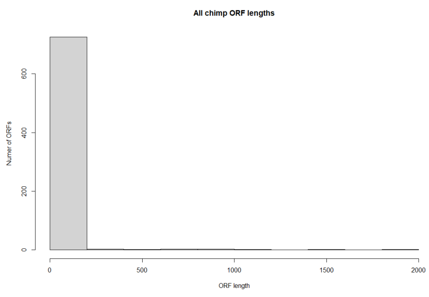


(a) All ORFs in human mtDNA

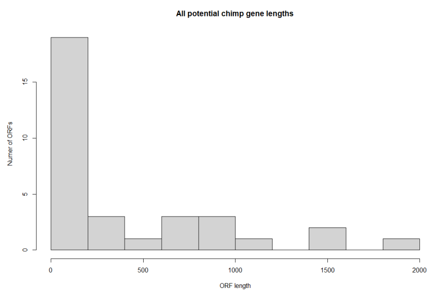


(b) Candidate protein coding genes in chimp

Figure 2 – Histograms showing the ORFs found in human mtDNA as well as the lengths of the candidate protein coding genes

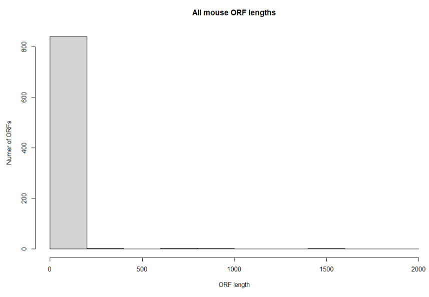


(a) All ORFs in chimp mtDNA

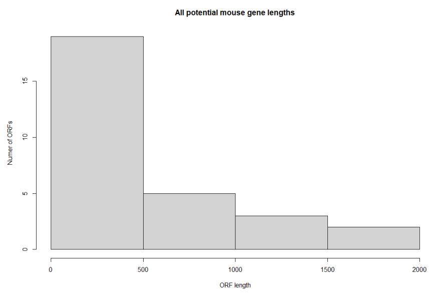


(b) Candidate protein coding genes in chimp

Figure 3 – Histograms showing the ORFs found in chimp mtDNA as well as the lengths of the candidate protein coding genes.



(a) All ORFs in mouse mtDNA



(b) Candidate protein coding genes in mouse

Figure 4 – Histograms showing the ORFs found in mouse mtDNA as well as the lengths of the candidate protein coding genes.

3 Chapter 3

3.1 Exercise 1

3.1.1 Background

It is often of interest to know how similar two DNA sequences are to each other. This similarity can then give hints about how closely related two sequences are and if they have a common ancestor. But with the large number of possible alignments, this task is not trivial, but with modern technology it is possible. There exists algorithms that can find both the best local alignment as well as the global alignment of two DNA sequences.

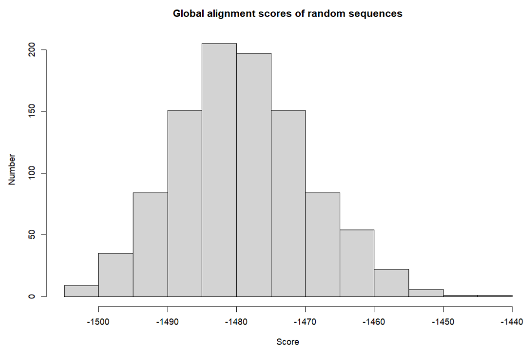
In this exercise the local and global alignments of the *D.melanogaster* gene *eyeless*(X79493) and the human gene *aniridia*(AY707088) has been investigated for their statistical significance.

3.1.2 What was done and discussion

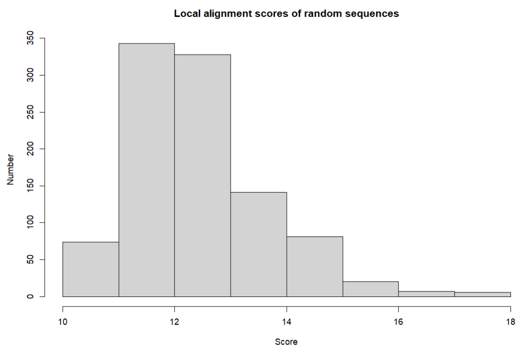
In this exercise the eyeless gene and the aniridia gene was downloaded from GenBank. They were then analyzed by using the programming language R. First the R-package Biostrings was used to convert the downloaded FASTA files to strings. Then a local and a global alignment was done. The gap opening and gap extension penalty was 3 and 1 respectively, and the substitution matrix used for the alignment looks like this:

	A	C	G	T
A	1	-1	-1	-1
C	-1	1	-1	-1
G	-1	-1	1	-1
T	-1	-1	-1	1

The resulting score for the local alignment was 183 for the score of the global alignment was -1204.
Next, the statistical significance was tested by creating 1000 random sequences of the same length as both the *eyeless* gene and the *aniridia*. Both a local and global alignment was done on all randomly generated sequences. The resulting scores can be seen in figure 5. Comparing the resulting scores from the random sequences and the score from the actual alignments we can be seen that the following: For the local alignment, the highest score obtained in the random sequences is 18, and the actual alignment score was 183, which is about a 10 times higher score, therefore the chance that this high of a score is very unlikely to be obtained just by chance. Similarly for the global alignment the highest score for a random sequence was -1442, while the score for the actual alignment was -1204, which is quite a bit higher than the random sequences. Therefore this score is also very unlikely to be this high or higher just by chance.
To view the code for this exercise, see Appendix A.C



(a) Global alignment of 1000 randomly generated sequences



(b) Local alignment of 1000 randomly generated sequences

Figure 5

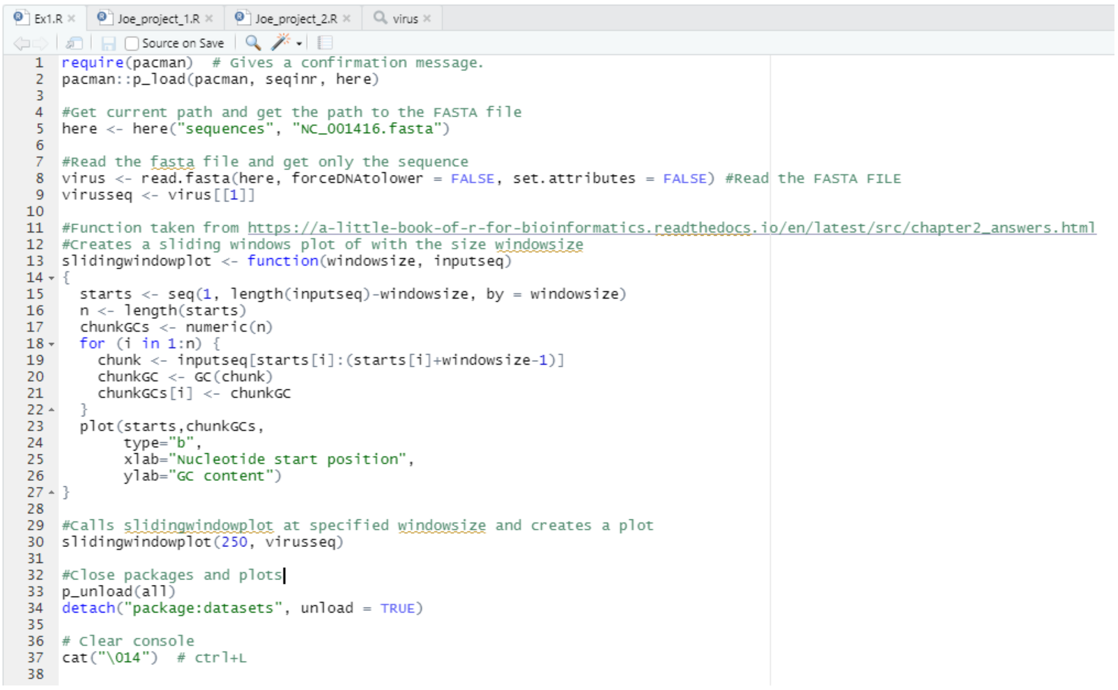
A few different approaches could have been taken when solving this exercise. The largest improvement could potentially have been to use a more biologically-based transition matrix, like a BLOSUM or PAM matrix, as well as different values for the gap opening and gap extension penalties. I was however unsure of how far apart it is thought that these gene sequences are from each other, so I decided to just use a standard and basic transition matrix and gap penalties as to not use a matrix or values that captures some other biological factors that I was not interested in this exercise.
When generating the random sequences I chose to generate all of them to exactly the same length as both the *eyeless* and *aniridia* genes, and with a 25% chance for all nucleotides at each position. It may have given a more accurate result if the nucleotide percentages reflected more accurately the actual makeup of the genes, but I chose to continue using just a very simple model for this exercise.

Appendices

A R-code

The sections below shows the R-code used to solve the exercises in this report. All code can also be found at this github-link: `https://github.com/AlfonsEdbom/Exercises_Statistics_in_Genetics.git`

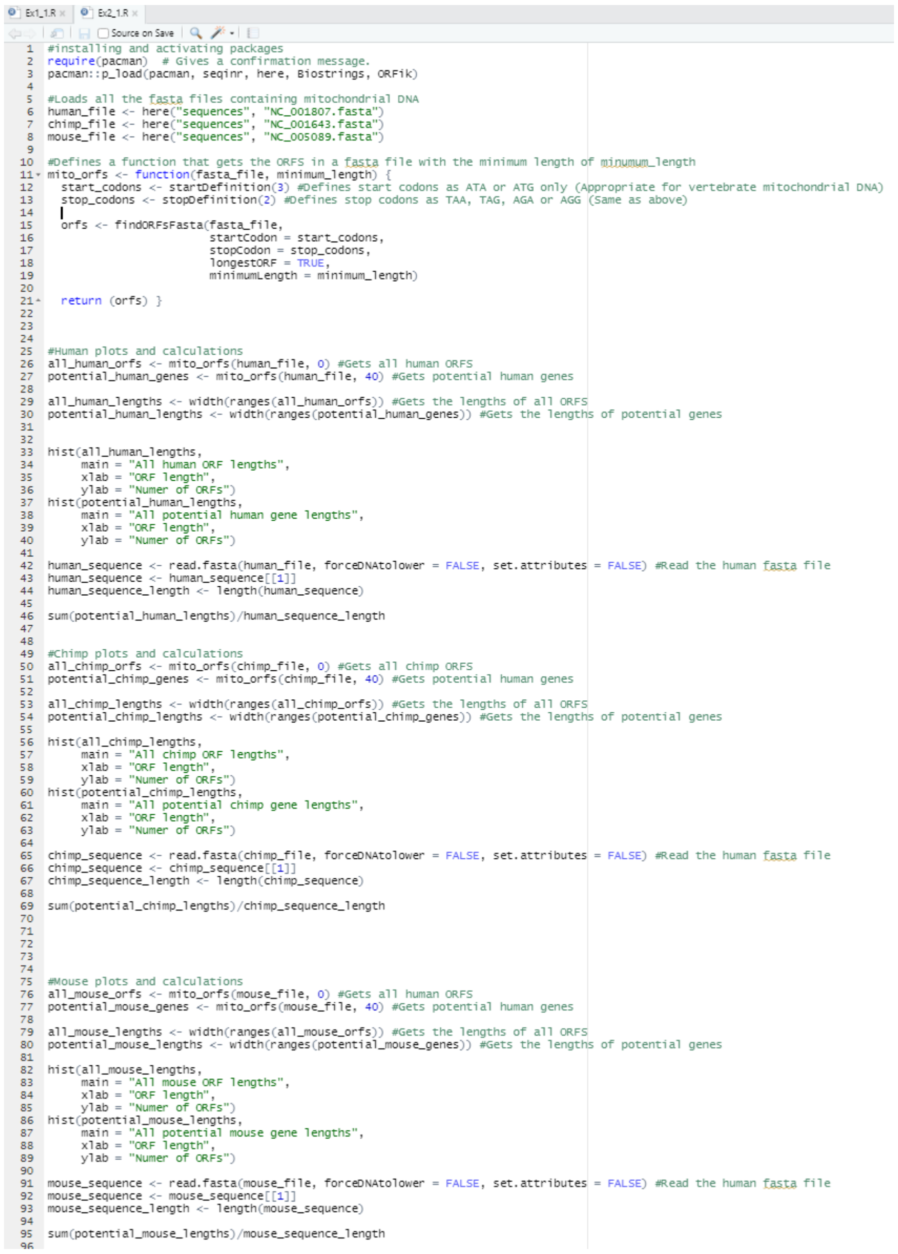
A.A Exercise 1.1



```r
require(pacman)  # Gives a confirmation message.
pacman::p_load(pacman, seqinr, here)

#Get current path and get the path to the FASTA file
here <- here("sequences", "NC_001416.fasta")

#Read the fasta file and get only the sequence
virus <- read.fasta(here, forceDNAtolower = FALSE, set.attributes = FALSE) #Read the FASTA FILE
virusseq <- virus[[1]]

#Function taken from https://a-little-book-of-r-for-bioinformatics.readthedocs.io/en/latest/src/chapter2_answers.html
#Creates a sliding windows plot of with the size windowsize
slidingwindowplot <- function(windowsize, inputseq)
{
  starts <- seq(1, length(inputseq)-windowsize, by = windowsize)
  n <- length(starts)
  chunkGCs <- numeric(n)
  for (i in 1:n) {
    chunk <- inputseq[starts[i]:(starts[i]+windowsize-1)]
    chunkGC <- GC(chunk)
    chunkGCs[i] <- chunkGC
  }
  plot(starts,chunkGCs,
       type="b",
       xlab="Nucleotide start position",
       ylab="GC content")
}

#Calls slidingwindowplot at specified windowsize and creates a plot
slidingwindowplot(250, virusseq)

#Close packages and plots
p_unload(all)
detach("package:datasets", unload = TRUE)

# Clear console
cat("\014")  # ctrl+L

```

Figure 6 – The code used to create sliding window plots in Exercise 1.1

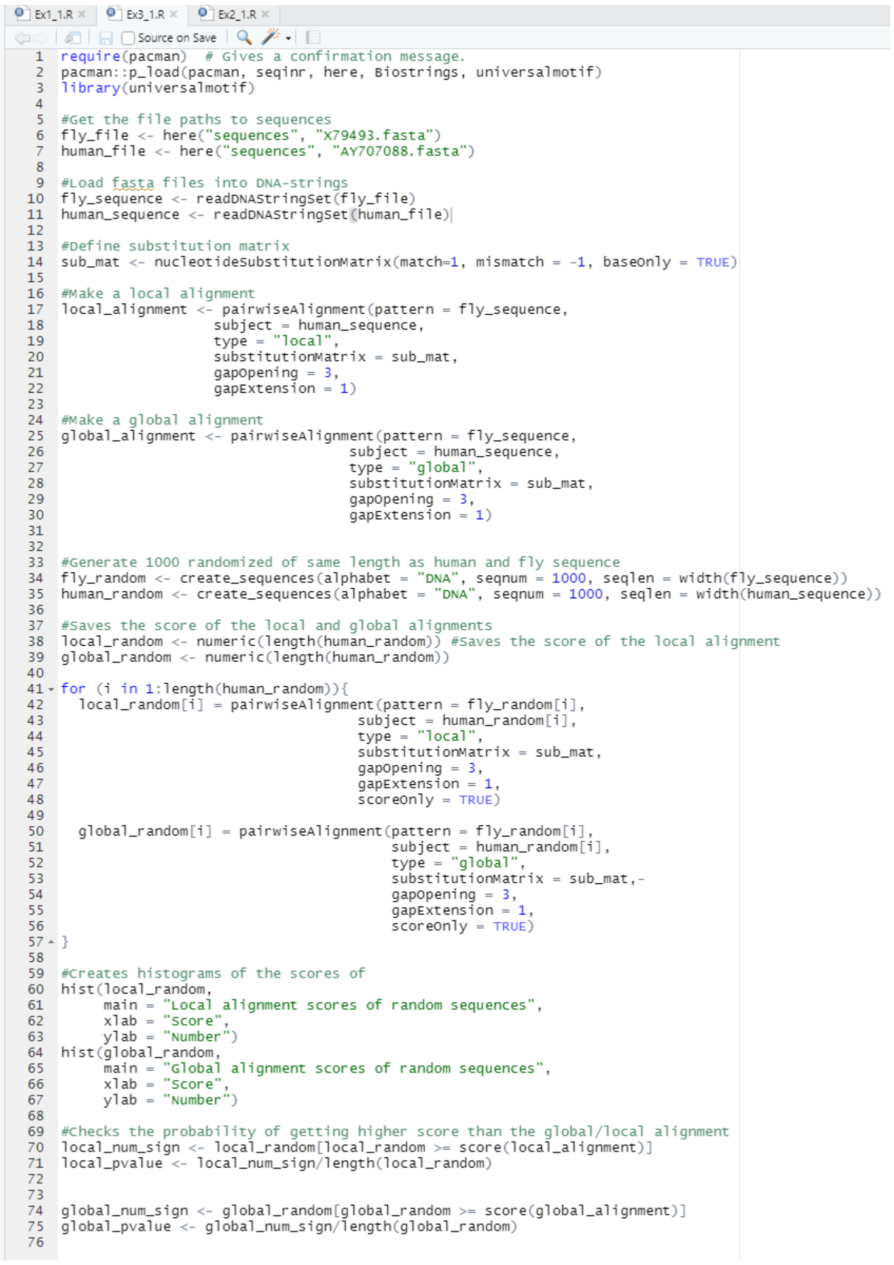
A.B Exercise 2.1



```r
#installing and activating packages
require(pacman)  # Gives a confirmation message.
pacman::p_load(pacman, seqinr, here, Biostrings, ORFik)

#Loads all the fasta files containing mitochondrial DNA
human_file <- here("sequences", "NC_001807.fasta")
chimp_file <- here("sequences", "NC_001643.fasta")
mouse_file <- here("sequences", "NC_005089.fasta")

#Defines a function that gets the ORFS in a fasta file with the minimum length of minumum_length
mito_orfs <- function(fasta_file, minimum_length) {
  start_codons <- startDefinition(3) #Defines start codons as ATA or ATG only (Appropriate for vertebrate mitochondrial DNA)
  stop_codons <- stopDefinition(2) #Defines stop codons as TAA, TAG, AGA or AGG (Same as above)

  orfs <- findORFsFasta(fasta_file,
                       startCodon = start_codons,
                       stopCodon = stop_codons,
                       longestORF = TRUE,
                       minimumLength = minimum_length)

  return (orfs) }



#Human plots and calculations
all_human_orfs <- mito_orfs(human_file, 0) #Gets all human ORFS
potential_human_genes <- mito_orfs(human_file, 40) #Gets potential human genes

all_human_lengths <- width(ranges(all_human_orfs)) #Gets the lengths of all ORFS
potential_human_lengths <- width(ranges(potential_human_genes)) #Gets the lengths of potential genes


hist(all_human_lengths,
     main = "All human ORF lengths",
     xlab = "ORF length",
     ylab = "Numer of ORFs")
hist(potential_human_lengths,
     main = "All potential human gene lengths",
     xlab = "ORF length",
     ylab = "Numer of ORFs")

human_sequence <- read.fasta(human_file, forceDNAtolower = FALSE, set.attributes = FALSE) #Read the human fasta file
human_sequence <- human_sequence[[1]]
human_sequence_length <- length(human_sequence)

sum(potential_human_lengths)/human_sequence_length


#Chimp plots and calculations
all_chimp_orfs <- mito_orfs(chimp_file, 0) #Gets all chimp ORFS
potential_chimp_genes <- mito_orfs(chimp_file, 40) #Gets potential human genes

all_chimp_lengths <- width(ranges(all_chimp_orfs)) #Gets the lengths of all ORFS
potential_chimp_lengths <- width(ranges(potential_chimp_genes)) #Gets the lengths of potential genes

hist(all_chimp_lengths,
     main = "All chimp ORF lengths",
     xlab = "ORF length",
     ylab = "Numer of ORFs")
hist(potential_chimp_lengths,
     main = "All potential chimp gene lengths",
     xlab = "ORF length",
     ylab = "Numer of ORFs")

chimp_sequence <- read.fasta(chimp_file, forceDNAtolower = FALSE, set.attributes = FALSE) #Read the human fasta file
chimp_sequence <- chimp_sequence[[1]]
chimp_sequence_length <- length(chimp_sequence)

sum(potential_chimp_lengths)/chimp_sequence_length





#Mouse plots and calculations
all_mouse_orfs <- mito_orfs(mouse_file, 0) #Gets all human ORFS
potential_mouse_genes <- mito_orfs(mouse_file, 40) #Gets potential human genes

all_mouse_lengths <- width(ranges(all_mouse_orfs)) #Gets the lengths of all ORFS
potential_mouse_lengths <- width(ranges(potential_mouse_genes)) #Gets the lengths of potential genes

hist(all_mouse_lengths,
     main = "All mouse ORF lengths",
     xlab = "ORF length",
     ylab = "Numer of ORFs")
hist(potential_mouse_lengths,
     main = "All potential mouse gene lengths",
     xlab = "ORF length",
     ylab = "Numer of ORFs")

mouse_sequence <- read.fasta(mouse_file, forceDNAtolower = FALSE, set.attributes = FALSE) #Read the human fasta file
mouse_sequence <- mouse_sequence[[1]]
mouse_sequence_length <- length(mouse_sequence)

sum(potential_mouse_lengths)/mouse_sequence_length
```

Figure 7 – The code used to solve Exercise 2.1

A.C Exercise 3.1



```r
require(pacman)  # Gives a confirmation message.
pacman::p_load(pacman, seqinr, here, Biostrings, universalmotif)
library(universalmotif)

#Get the file paths to sequences
fly_file <- here("sequences", "X79493.fasta")
human_file <- here("sequences", "AY707088.fasta")

#Load fasta files into DNA-strings
fly_sequence <- readDNAStringSet(fly_file)
human_sequence <- readDNAStringSet(human_file)

#Define substitution matrix
sub_mat <- nucleotideSubstitutionMatrix(match=1, mismatch = -1, baseOnly = TRUE)

#Make a local alignment
local_alignment <- pairwiseAlignment(pattern = fly_sequence,
                  subject = human_sequence,
                  type = "local",
                  substitutionMatrix = sub_mat,
                  gapOpening = 3,
                  gapExtension = 1)

#Make a global alignment
global_alignment <- pairwiseAlignment(pattern = fly_sequence,
                                  subject = human_sequence,
                                  type = "global",
                                  substitutionMatrix = sub_mat,
                                  gapOpening = 3,
                                  gapExtension = 1)


#Generate 1000 randomized of same length as human and fly sequence
fly_random <- create_sequences(alphabet = "DNA", seqnum = 1000, seqlen = width(fly_sequence))
human_random <- create_sequences(alphabet = "DNA", seqnum = 1000, seqlen = width(human_sequence))

#Saves the score of the local and global alignments
local_random <- numeric(length(human_random)) #Saves the score of the local alignment
global_random <- numeric(length(human_random))

for (i in 1:length(human_random)){
  local_random[i] = pairwiseAlignment(pattern = fly_random[i],
                                   subject = human_random[i],
                                   type = "local",
                                   substitutionMatrix = sub_mat,
                                   gapOpening = 3,
                                   gapExtension = 1,
                                   scoreOnly = TRUE)

  global_random[i] = pairwiseAlignment(pattern = fly_random[i],
                                       subject = human_random[i],
                                       type = "global",
                                       substitutionMatrix = sub_mat,-
                                       gapOpening = 3,
                                       gapExtension = 1,
                                       scoreOnly = TRUE)
}

#Creates histograms of the scores of
hist(local_random,
     main = "Local alignment scores of random sequences",
     xlab = "Score",
     ylab = "Number")
hist(global_random,
     main = "Global alignment scores of random sequences",
     xlab = "Score",
     ylab = "Number")

#Checks the probability of getting higher score than the global/local alignment
local_num_sign <- local_random[local_random >= score(local_alignment)]
local_pvalue <- local_num_sign/length(local_random)


global_num_sign <- global_random[global_random >= score(global_alignment)]
global_pvalue <- global_num_sign/length(global_random)
```

Figure 8 – The code used to solve Exercise 3.1