Lab 2 - Gr. 14 - Bioinformatics (732A93)

Andreas Stasinakis (andst745), Hector Plata (hecpl268), Julius Kittler (julki092), Mim Kemal Tekin (mimte666), Stefano Toffol (steto820)

Assignment 1

Question 1.1

Starting from 33 DNA sequence of various species of casque-headed lizard (Basiliscus basiliscus), other 33 sequences of nucleotides have been generated. The sampling probabilities are the same of the real proportions of the original dataset.
After the artificial DNA has been created, the base frequencies are compared in Table 1. As expected, the observed proportions of the generated data closely resamble the theoretical ones.

Question 1.2

- Created one phylogenetic tree with 33 tips
- For each original DNA sequence of the 33 available, used the function `simSeq(.)` from package `phangorn` to simulate the sequences.
- Result: 33 phylogenetic tree, one for DNA sequence, each with 33 tips.

Is the result Krzysztof wanted? I've read the suggested materials and the lecture slides, but I am still not sure how to use the tree in order to simulate the sequences.

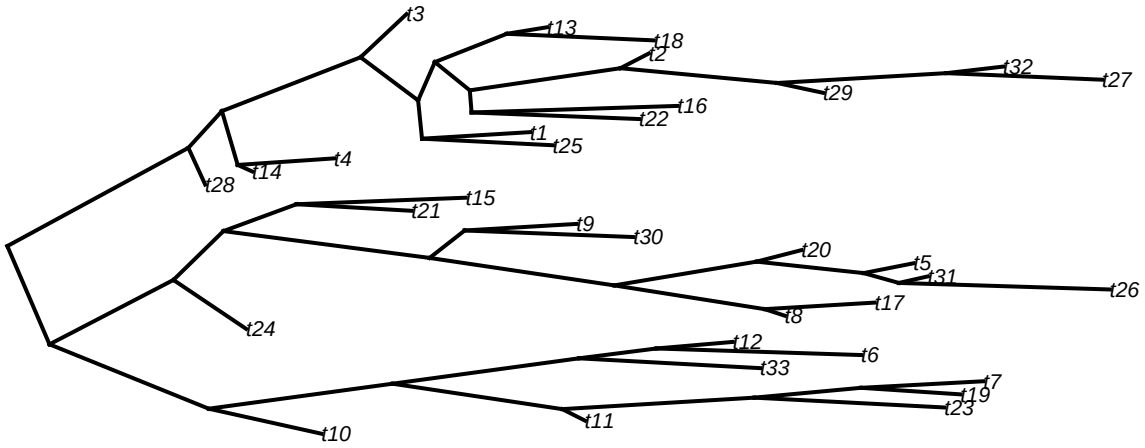

Table 1: Base frequencies of the 33 original and generated DNA sequences.

Base	Original frequency	Simulated frequency
a	0.3121	0.3120
c	0.2052	0.2045
g	0.2307	0.2329
t	0.2519	0.2505

Table 2: Base frequencies of the 33 original DNA sequences and of the 33 simulated phylogenetic trees.

Base	Original frequency	Simulated frequency
a	0.3121	0.3126
c	0.2052	0.2053
g	0.2307	0.2307
t	0.2519	0.2514

Table 3: Lenght of the 33 original DNA sequences and of the 33 simulated phylogenetic trees.

Base	Original frequency	Simulated frequency
1	997	1012
2	2709	2597
3	2891	2634
4	1006	1061
5	1474	1369
6	1091	1059
7	2891	2762
8	1004	904
9	1044	1068
10	2920	3174
11	2816	2958
12	2886	3135
13	2748	2807
14	2835	2731
15	2744	2629
16	1003	965
17	999	1023
18	2825	2977
19	2845	2573
20	1008	1068
21	1472	1380
22	2892	2945
23	1444	1350
24	1000	936
25	2837	2940
26	2506	2623
27	1005	1073
28	2886	2772
29	2890	2683
30	1060	1117
31	2737	2993
32	1003	1027
33	931	983

Assignment 2

Question 2.1

1. Some basic statistics on each sequence dataset:

Table 4: Base composition

	Original	Simulated_Sample	Simulated_Tree
a	0.3121454	0.3120196	0.3125979
c	0.2052325	0.2045236	0.2052663
g	0.2307222	0.2329182	0.2307187
t	0.2518999	0.2505387	0.2514171

Table 5: Percentage of G + C

Original	Simulated_Sample	Simulated_Tree
0.4359547	0.4374417	0.435985

Table 6: Percentage of A + T

Original	Simulated_Sample	Simulated_Tree
0.5640453	0.5625583	0.564015

2. Translate nucleotid sequences into protein sequences & report amino acid composition:

Table 7: Amino Acid Composition (in %)

	Name	Original	Simulated_Sample	Simulated_Tree
*	Stp	0.0582622	0.0604643	0.0502917
A	Ala	0.0417469	0.0504175	0.0397463
C	Cys	0.0435361	0.0290852	0.0206733
D	Asp	0.0367924	0.0311038	0.0205467
E	Glu	0.0603725	0.0401872	0.0340046
F	Phe	0.0378475	0.0298651	0.0553904
G	Gly	0.0501881	0.0545922	0.0869681
H	His	0.0348656	0.0304615	0.0183605
I	Ile	0.0369759	0.0601431	0.0498979
K	Lys	0.0674833	0.0528030	0.0947262
L	Leu	0.0864758	0.0829893	0.0845023
M	Met	0.0182586	0.0181668	0.0124101
N	Asn	0.0338563	0.0453253	0.0552847
P	Pro	0.0578952	0.0410588	0.0833236
Q	Gln	0.0461510	0.0358749	0.0277606
R	Arg	0.0637673	0.0853748	0.0730704
S	Ser	0.0901918	0.0821635	0.0575919
T	Thr	0.0501422	0.0634921	0.0455436
V	Val	0.0434902	0.0578035	0.0490824

	Name	Original	Simulated_Sample	Simulated_Tree
W	Trp	0.0186256	0.0139462	0.0158988
Y	Tyr	0.0221580	0.0346821	0.0249260
X	gaps	0.0009175	0.0000000	0.0000000

3. Obtain number of stop codons in simulated sequences & compare to true seq:

There are 1270 stop codons in the original sequence, 1298 stop codons in the simulated sequence without and 1333 stop codons in the simulated sequence with
trees.

Table 8: Number of Stop Codons

	*
Original	1270
Simulated_Sample	1318
Simulated_Tree	36140

Question 2.2

Expected Markov chain order:

I would expect order of 2 (or more) for the original data sets because three nucleotids code for an amino-acid and since the amino-acids are not fully independent in the DNA. Note that the number of required free parameters increases a lot from 2 to 3 (see below). The larger the number of parameters, the less reliable the estimated probabilities for the transition matrix will be.

For the simulated data sets, I would expect order of 1 because here, the nucleotids are indeed random and should therefore be independent.

Number of free parameters required for order of 1:

$$4 * (4 - 1) = 12$$

Number of free parameters required for order of 2:

$$4^2 * (4^2 - 1) = 240$$

Number of free parameters required for order of 3:

$$4^3 * (4^3 - 1) = 4032$$

Actual number of nucleotids by dataset:

```
> length(unlist(original))
[1] 65435
> length(unlist(sim_sample))
[1] 65435
> length(unlist(sim_tree))
[1] 66000
```

First, we concatenate the sequences from the three data sets and remove any characters that are not "a", "c", "g", "t". **Second**, we conduct a Chi-Square test with the H0: Sequence is of 1st order. The H0 get's rejected for the original sequence (since $p = 0$) but not for the two sampled sequences (as $p > 0.05$). This is what we expected: First order Markov chains for the randomly sampled sequence but second or higher order Markov chains for the original sequence (in which dependency exists).

```
## Assess the order with Chi-Square test:

## Original:

## The assessOrder test statistic is:  1302.385  the Chi-Square d.f. are:  36  the p-value is:  0

##
## -------------

## Simulated_Sample:

## The assessOrder test statistic is:  24.46538  the Chi-Square d.f. are:  36  the p-value is:  0.92767

##
## -------------

## Simulated_Tree:

## The assessOrder test statistic is:  13675.14  the Chi-Square d.f. are:  36  the p-value is:  0

##
## -------------
```

Third, we fit 1st order Markov chains for all three data sets. Note that we tried to fit a higher order Markov chain for the original data set (since this would be appropriate based on the Chi-Square-Test). However, the function fitHigherOrder from the markovchain package did not produce correct transition matrices and we also did not find any other package that could fit a higher order Markov chain correctly. Therefore, we also fit a 1st order Markov chain for the original data set.

```
## Original
##  A  4 - dimensional discrete Markov Chain defined by the following states:
##  a, c, g, t
##  The transition matrix  (by rows)  is defined as follows:
##           a         c          g         t
## a 0.3378239 0.1733209 0.27506981 0.2137853
## c 0.3793026 0.2478021 0.05029057 0.3226047
## g 0.3937305 0.2032607 0.19338591 0.2096229
## t 0.1509045 0.2119097 0.35698677 0.2801991

##
## -------------

## Simulated_Sample
##  A  4 - dimensional discrete Markov Chain defined by the following states:
##  a, c, g, t
##  The transition matrix  (by rows)  is defined as follows:
##           a         c         g         t
## a 0.3122551 0.2044475 0.2327586 0.2505388
## c 0.3213779 0.2014496 0.2317866 0.2453859
## g 0.3081163 0.2106817 0.2312840 0.2499180
## t 0.3076735 0.2014152 0.2355740 0.2553373
```

```
##
## -------------
## Simulated_Tree
##  A  4 - dimensional discrete Markov Chain defined by the following states:
##  a, c, g, t
##  The transition matrix  (by rows)  is defined as follows:
##           a         c         g         t
## a 0.4401674 0.1597243 0.1878202 0.2122881
## c 0.2424195 0.3590136 0.2041205 0.1944463
## g 0.2550338 0.1811902 0.3414068 0.2223692
## t 0.2641072 0.1584599 0.2041970 0.3732360
```

Lastly, we consider the assumptions behind the analyses (see book "Statistical Methods in Bioinformatics"):

1. *Markov property*: "current state is all that matters in determining the probabilities for the states that the process will occupy in the future.
2. *Temporally homogenous transition probabilities property*: given that at time t, the process is in state E_j, the probability that one time unit later it is in state E_k is independent of t.

We consider nucleotid sequences here.

1. The *Markov property* may be violated, especially in the original sequence because the nucleotids code for amino-acids that will later on be translated into proteins. Therefore, one cannot actually assume that the previous nucleotids don't matter in determining the probabilities for the next states.
2. Regarding the second property, we don't consider time here but space (i.e. position of the nucleotids). From my point of view, it is a valid assumption that the position does not matter (however the type of the previous nucleotids does, see 1.).

Question 2.3

1. Choose a distance measure between sequences.
2. Calculate for each alignment the distances between all pairs of sequences.
3. Plot heatmaps visualizing the distances.
4. Comment on what you can observe.

Question 3

Question 3.1

Question 3.2

Appendix

```
knitr::opts_chunk$set(fig.width = 7, fig.height = 3, echo = FALSE,
                      warning = FALSE, message = FALSE)

library(dplyr)
library(tidyr)
```

```r
library(magrittr)
library(ape)          # General R-package for phylogenetics & comparative methods
library(seqinr)       # Specialized package for nucleotide sequence management
library(phangorn)
library(markovchain)  # For fitting, evaluating markov chains (question 2)

source("732A51_BioinformaticsHT2018_Lab02_GenBankGetCode.R")
# ------------------------------------------------------------------------------
# Question 1.1
# ------------------------------------------------------------------------------

lizards_format_sequences <- read.fasta(file = "lizard_seqs.fasta")  # Alternative version of the file
# Useful in some ways?

n <- length(lizards_accession_numbers)  # Number of sequences to reproduce
p <- base.freq(lizards_sequences)  # Probability of the base sequences
simulated_lizards <- list()  # Object that will contain our simulated data

# The names of the simulated data are the original names + "_sim"
# NOTE: it does not follow the format from GenBank
simulated_names <- paste(lizards_accession_numbers, "_sim", sep = "")

set.seed(1535)  # Set seed in order to reproduce the experiment
for(i in 1:n) {  # Cycle through every single object of the lizard_sequences
  len_seq <- length(lizards_sequences[[i]])  # Lenght of each sequence
  simulated_lizards[[ simulated_names[i] ]] <-
    sample(c("a", "c", "g", "t"), len_seq, replace = T, prob = p)
  # Creating the artificial sequence sampling with probabilities p equal to the original ones
  # NOTE: we use the general distribution for every single sequence
}

# Save as fasta file
write.dna(simulated_lizards, file = "simulated_lizards.fasta", format = "fasta", append = F,
          nbcol = 6, colsep = " ", colw = 10)

# Table with simulated base frequency
df_table <- data.frame("Base" = c("a", "c", "g", "t"),
                       "Original\nfrequency" = p,
                       "Simulated\nfrequency" = base.freq(as.DNAbin(simulated_lizards)),
                       row.names = NULL)
knitr::kable(df_table, booktabs = T, align = c("r", "l", "l"), digits = c(NA, 4, 4),
      col.names = c("Base", "Original\nfrequency", "Simulated\nfrequency"), format = "latex",
      caption = "Base frequencies of the 33 original and generated DNA sequences.")
# ------------------------------------------------------------------------------
# Question 1.2
# ------------------------------------------------------------------------------

##############################################################################
# Create the tree
##############################################################################

# Set the random seed for reproducibility once again
set.seed(1545)
```

```r
# Simulate the tree: it has to be with 33 tips. We try to generate a tree with uniform distribution
# (default function)
simulated_tree <- rtree(33)

# Plot the tree:
par(lend = 2)
par(mar = rep(2, 4))
par(cex = 0.8)

plot(simulated_tree, type = "cladogram", edge.width = 2)  # Cladogram template, the easiest to
                                                          # understand for this tree


################################################################################
# Simulate the sequences
################################################################################

# Object where to store the randomic sequences
simulated_tree_seq <- list()
# The names of the simulated data are the original names + "_sim_tree"
# NOTE: it does not follow the format from GenBank
simulated_names_tree <- paste(lizards_accession_numbers, "_sim_tree", sep = "")


# We now define a function to compute a partially random transiction matrix:
# It first calculate first-order Markov transition matrix where each *row*
# corresponds to a single run of the Markov chain. It then add some random noise.

trans.matrix <- function(DNA_seq) {

  # Retrieve unique elements
  DNA_unique <- unique(DNA_seq)

  # Create an empty matrix
  matrix <- matrix(0, ncol = length(DNA_unique), nrow=length(DNA_unique))

  # Fill it: to count i and element i + 1 and add one in the corresponding cell of the matrix.
  for (i in 1:(length(DNA_seq) - 1)) {
    index_of_i <- DNA_unique == DNA_seq[i]
    index_of_i_plus_1 <- DNA_unique == DNA_seq[i + 1]
    matrix[index_of_i, index_of_i_plus_1] = matrix[index_of_i, index_of_i_plus_1] + 1
  }

  # Normalize it
  matrix <- matrix / rowSums(matrix)

  # Add random noise, keeping the rowSum equal to 1
  # Proceed row by row
  for(i in nrow(matrix)) {
    noise <- rnorm(1, mean = 0,  sd = 0.01)   # Random quantity to ADD/SUBtract
    ind_add <- floor(runif(1, 1, nrow(matrix)+1))  # Column index where to add
    ind_sub <- floor(runif(1, 1, nrow(matrix)+1))  # Column index where to subtract
    matrix[i,ind_add] <- matrix[i,ind_add] + noise  # Add noise to one prob...
```

```
    matrix[i,ind_sub] <- matrix[i,ind_sub] - noise  # ...and subtract from another
  }

  return(matrix)
}


# Time to generate! Set the seed again (better safe than sorry)
set.seed(1545)
# Here we will store all the newly generated lenght of the sequences
len_generated_sequences <- rep(NA, 33)
len_original_sequences <- rep(NA, 33)

# Create one sequence for each one of the originals
for(i in 1:n) {
  # Problem: original sequences include unknown nucleotides --> delete them
  known_nucleotides <- lizards_sequences[[i]][lizards_sequences[[i]] %in% c(18, 28, 48, 88)]

  # Decide lenght randomically: we take the lenght of the original known sequence +
  # uniform random number equal max to 10% of the original lenght
  len_seq <- length(known_nucleotides)
  len_original_sequences[i] <- len_seq
  len_seq <- len_seq + floor(runif(1, -floor(len_seq/10), floor(len_seq/10)))
  len_generated_sequences[i] <- len_seq

  # We re-use the same probabilities computed before, referring to the original dataset.
  # We also compute the transiction matrix using the function defined above.
  simulated_tree_seq[[ simulated_names_tree[i] ]] <-
    simSeq(simulated_tree, l = len_seq, bf = p, Q = trans.matrix(known_nucleotides))
}

# Convert the 33 generated trees as DNAbin
simulated_tree_seq2 <- as.DNAbin(simulated_tree_seq)

# Save as fasta file
write.dna(simulated_tree_seq2, file = "simulated_lizards_tree.fasta", format = "fasta", append = F,
          nbcol = 6, colsep = " ", colw = 10)

# Table with simulated base frequency
df_table <- data.frame("Base" = c("a", "c", "g", "t"),
                       "Original\nfrequency" = p,
                       "Simulated\nfrequency" = base.freq(simulated_tree_seq2),
                       row.names = NULL)

knitr::kable(df_table, booktabs = T, align = c("r", "l", "l"), digits = c(NA, 4, 4),
      col.names = c("Base", "Original\nfrequency", "Simulated\nfrequency"), format = "latex",
      caption = "Base frequencies of the 33 original DNA sequences and of the 33 simulated phylogenetic
# Table with simulated lenght vs original
df_table <- data.frame("Sequence n." = 1:33,
                       "Original\nlenght" = len_original_sequences,
                       "Simulated\nlenght" = len_generated_sequences,
                       row.names = NULL)
```

```r
knitr::kable(df_table, booktabs = T, align = c("r", "l", "l"),
      col.names = c("Base", "Original\nfrequency", "Simulated\nfrequency"), format = "latex",
      caption = "Lenght of the 33 original DNA sequences and of the 33 simulated phylogenetic trees.")
# ------------------------------------------------------------------------
# Question 2.1
# ------------------------------------------------------------------------

# First read in all the data again
original = ape::read.FASTA(file = "data/lizard_seqs.fasta", type = "DNA")
sim_sample = ape::read.FASTA(file = "data/simulated_lizards.fasta", type = "DNA")
sim_tree = ape::read.FASTA(file = "data/simulated_lizards_tree.fasta", type = "DNA")

# 1. Some basic statistics on each sequence dataset ----------------------------

# Individual base composition
knitr::kable(data.frame(Original = ape::base.freq(original),
             Simulated_Sample = ape::base.freq(sim_sample),
             Simulated_Tree = ape::base.freq(sim_tree)),
             caption = "Base composition")

# GC content
knitr::kable(data.frame(Original = ape::GC.content(original),
             Simulated_Sample = ape::GC.content(sim_sample),
             Simulated_Tree = ape::GC.content(sim_tree)),
             caption = "Percentage of G + C")

# AT content
knitr::kable(data.frame(Original = 1 - ape::GC.content(original),
             Simulated_Sample = 1 - ape::GC.content(sim_sample),
             Simulated_Tree = 1 - ape::GC.content(sim_tree)),
             caption = "Percentage of A + T")


# 2. Translate nucleotid sequences into protein sequences & report amino acid comp.

# Create function to obtain amino acid percentages from DNAbin
get_aa_comp = function(DNAbin, relative = TRUE){

  # Data conversion to obtain amino distribution

  original_amino = ape::trans(DNAbin) # Convert DNAbin to AAbin (amino acids)
  original_amino = as.character(original_amino) # convert AAbin to char list
  original_amino = unlist(original_amino) # convert char list to char vector

  # Report amino acid composition

  if (relative == TRUE) {
    # Compute percentages
    metric = as.numeric(table(original_amino)) / length(original_amino)
    names(metric) = names(table(original_amino))
  } else {
    # Compute counts
    metric = as.numeric(table(original_amino))
```

```
    names(metric) = names(table(original_amino))
  }

  return(metric)
}

# Obtain percentages by amino acid precentages
Simulated_Sample = get_aa_comp(sim_sample)
Simulated_Tree = get_aa_comp(sim_tree)
Original = get_aa_comp(original)

# Original has an X at second to last position, we need to account for that
# According to docu of ape::trans, this is because of alignment gaps when
# e.g. the sequence does not have full 3 proteins.
Simulated_Sample = c(Simulated_Sample, X = 0)
Simulated_Tree = c(Simulated_Tree, X = 0)
Original = Original[c(1:(length(Original)-2), length(Original), length(Original)-1)]

knitr::kable(data.frame(Name = c(seqinr::aaa(), "gaps"),
                        Original,
                        Simulated_Sample,
                        Simulated_Tree),
             caption = "Amino Acid Composition (in %)")


# 3. Obtain number of stop codons in simulated sequences & compare to true seq

Simulated_Sample = get_aa_comp(sim_sample, relative = FALSE)
Simulated_Tree = get_aa_comp(sim_tree, relative = FALSE)
Original = get_aa_comp(original, relative = FALSE)

knitr::kable(t(data.frame(Original = Original["*"],
               Simulated_Sample = Simulated_Sample["*"],
               Simulated_Tree = Simulated_Tree["*"])),
             caption = "Number of Stop Codons")


# ------------------------------------------------------------------------------
# Question 2.2
# ------------------------------------------------------------------------------

# 1. Concatenate the sequences from the three data sets and remove any
# characters that are not "a", "c", "g", "t"

original_seq = unlist(as.character(original)) # obtain character vector
original_seq = original_seq[original_seq %in% c("a", "c", "g", "t")] # a, c, g, t

sim_sample_seq = unlist(as.character(sim_sample)) # obtain character vector
sim_sample_seq = sim_sample_seq[sim_sample_seq %in% c("a", "c", "g", "t")] # a, c, g, t

sim_tree_seq = unlist(as.character(sim_tree)) # obtain character vector
sim_tree_seq = sim_tree_seq[sim_tree_seq %in% c("a", "c", "g", "t")] # a, c, g, t
```

```r
# 2. Assess the order with Chi-Square test

# This test returns a list of the chi-squared value and the p-value.
# If the p-value is greater than the given significance level, we cannot reject
# the hypothesis that the sequence is of first order.

cat("Assess the order with Chi-Square test:\n")
cat("Original:\n"); assessOrder(original_seq); cat("\n-------------\n")
cat("Simulated_Sample:\n"); assessOrder(sim_sample_seq); cat("\n-------------\n")
cat("Simulated_Tree:\n"); assessOrder(sim_tree_seq); cat("\n-------------\n")

# 3. Fit markov chains

# 1st order markov chains for sampled data ------------------------------------

original_fit = markovchainFit(data = original_seq, confidencelevel = 0.95,
                              name = "Original")
original_fit$estimate; cat("\n-------------\n")

sim_sample_fit = markovchainFit(data = sim_sample_seq, confidencelevel = 0.95,
                              name = "Simulated_Sample")
sim_sample_fit$estimate; cat("\n-------------\n")


sim_tree_fit = markovchainFit(data = sim_tree_seq, confidencelevel = 0.95,
                              name = "Simulated_Tree")
sim_tree_fit$estimate


# Fitting higher order markov chains did not work
# original_fit_2nd = fitHigherOrder(original_seq, order = 2)
# cat("Original, 2nd order:\n\n"); original_fit_2nd; cat("\n-------------\n")

# ------------------------------------------------------------------------------
# Question 2.3
# ------------------------------------------------------------------------------

# ------------------------------------------------------------------------------
# Question 3.1
# ------------------------------------------------------------------------------

# ------------------------------------------------------------------------------
# Question 3.2
# ------------------------------------------------------------------------------
```