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

Julius Kittler (julki092), Stefano Toffol (steto820), Saewon Jun (saeju204), Maximilian Pfundstein (maxpf364)

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.

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

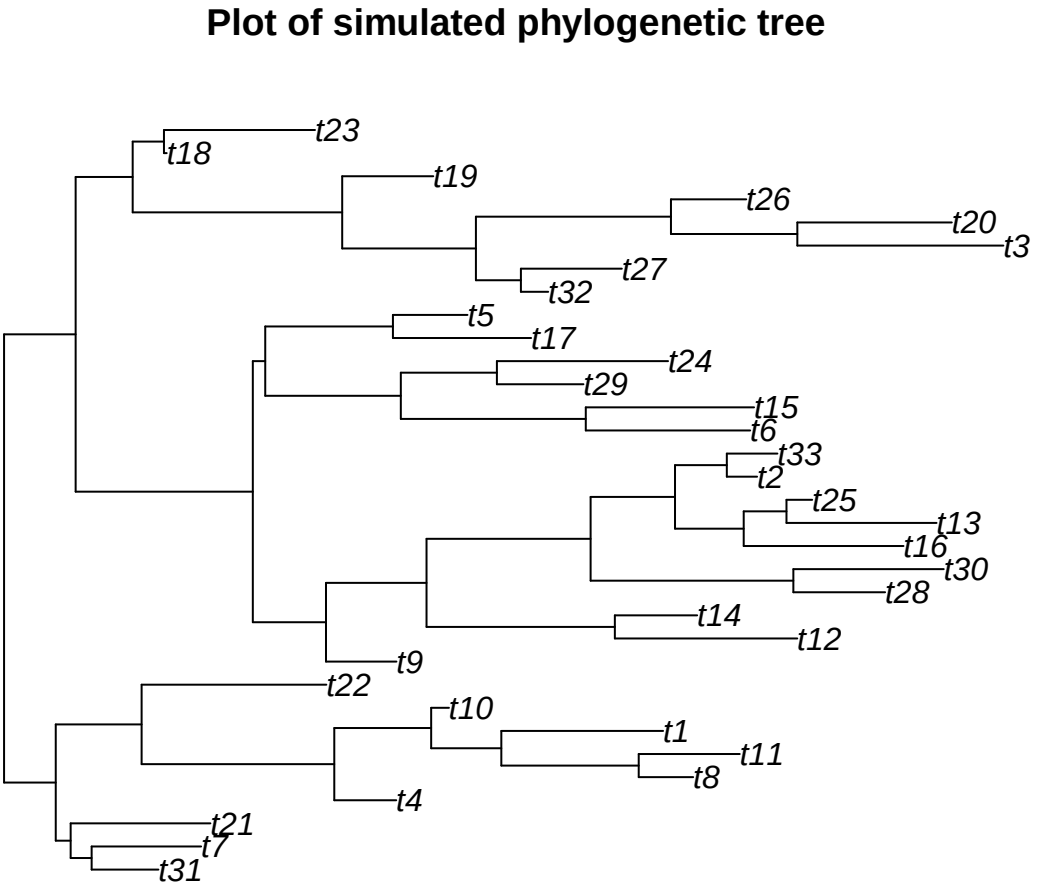
Base	Original frequency	Simulated frequency
a	0.3121	0.3120
c	0.2052	0.2045
g	0.2307	0.2329
t	0.2519	0.2505

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.

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	0.3150
c	0.2052	0.2061
g	0.2307	0.2258
t	0.2519	0.2530



Assignment 2

Question 2.1

1. Some basic statistics on each sequence dataset:

As you can be see below, base composition, G+C content and A+T content are very similar but not exactly the same across data sets (because of the random samples).

Table 3: Base composition

	Original	Simulated_Sample	Simulated_Tree
a	0.3121454	0.3120196	0.3150152
c	0.2052325	0.2045236	0.2061061
g	0.2307222	0.2329182	0.2258333
t	0.2518999	0.2505387	0.2530455

Table 4: Percentage of G + C

Original	Simulated_Sample	Simulated_Tree
0.4359547	0.4374417	0.4319394

Table 5: Percentage of A + T

Original	Simulated_Sample	Simulated_Tree
0.5640453	0.5625583	0.5680606

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

The overview of the amino-acid composition by data set can be seen in the table "Amino Acid Composition (in %)".

- *Overall:* Overall, the amino-acid compositions are similar. However, they can differ by up to ca. 3% by amino-acid.
- *Gaps:* Note that gaps ("X") were recognized for the original data set but not for the sampled data sets. According to the docu of ape::trans, this is because of alignment gaps when e.g. the sequence does not have full 3 proteins.

Table 6: Amino Acid Composition (in %)

	Name	Original	Simulated_Sample	Simulated_Tree
*	Stp	0.0582622	0.0604643	0.0606516
A	Ala	0.0417469	0.0504175	0.0478205
C	Cys	0.0435361	0.0290852	0.0254800
D	Asp	0.0367924	0.0311038	0.0323505
E	Glu	0.0603725	0.0401872	0.0384020
F	Phe	0.0378475	0.0298651	0.0292565
G	Gly	0.0501881	0.0545922	0.0475475
H	His	0.0348656	0.0304615	0.0307125
I	Ile	0.0369759	0.0601431	0.0593776
K	Lys	0.0674833	0.0528030	0.0551006
L	Leu	0.0864758	0.0829893	0.0887251
M	Met	0.0182586	0.0181668	0.0188825
N	Asn	0.0338563	0.0453253	0.0465465
P	Pro	0.0578952	0.0410588	0.0439985
Q	Gln	0.0461510	0.0358749	0.0334880
R	Arg	0.0637673	0.0853748	0.0882246
S	Ser	0.0901918	0.0821635	0.0849486
T	Thr	0.0501422	0.0634921	0.0628811
V	Val	0.0434902	0.0578035	0.0549186
W	Trp	0.0186256	0.0139462	0.0124670
Y	Tyr	0.0221580	0.0346821	0.0382200
X	gaps	0.0009175	0.0000000	0.0000000

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

The number of stop codons can be seen in the table "Number of Stop Codons". There are 1270 stop codons in the original sequence, 1318 stop codons in the simulated sequence without and 1333 stop codons in the simulated sequence with trees.

Table 7: Number of Stop Codons

	*
Original	1270
Simulated_Sample	1318
Simulated_Tree	1333

Question 2.2

Expected Markov chain order:

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

We 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. The number of nucleotids for the data sets are: 65435 for original data set and sampled data set and 66000 for the data set sampled from the trees. If you consider the number of parameters for the 1st to 3rd order below, it should be realistic to estimate parameters for at least 2nd order given the amount of data available.

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:

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

Assess Markov chain order:

First, we concatenate the sequences from the three data sets and remove any characters that are not “a”, “c”, “g”, “t”. Then, 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: p =  0

##
## Simulated_Sample: p = 0.9276794

##
## Simulated_Tree: p = 0.9823262
```

Fit Markov chains:

Lastly, we fit 1st order Markov chains for all three data sets. The corresponding three transition matrices can be found below.

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 0.3145262 0.2061087 0.2279942 0.2513709
## c 0.3175035 0.2039991 0.2243623 0.2541351
## g 0.3174103 0.2093257 0.2207313 0.2525327
## t 0.3114784 0.2049578 0.2289084 0.2546554
```

Now, 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). It is a valid assumption that the position does not matter (however the type of the previous nucleotids

does, see 1.).

Question 2.3

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

Clustal Omega (https://www.ebi.ac.uk/Tools/msa/clustalo/) was used for performing the multiple sequence alignment. An alternative would be using the package `DECIPHER` with the function `AlignSeqs`.

We used the function `dist.alignment` from the `seqinr` package for calculating the distances. The documentation states:

> These functions compute a matrix of pairwise distances from aligned sequencesusing similarity (Fitch matrix, for protein sequences only) or identity matrix (for protein and DNA sequences).

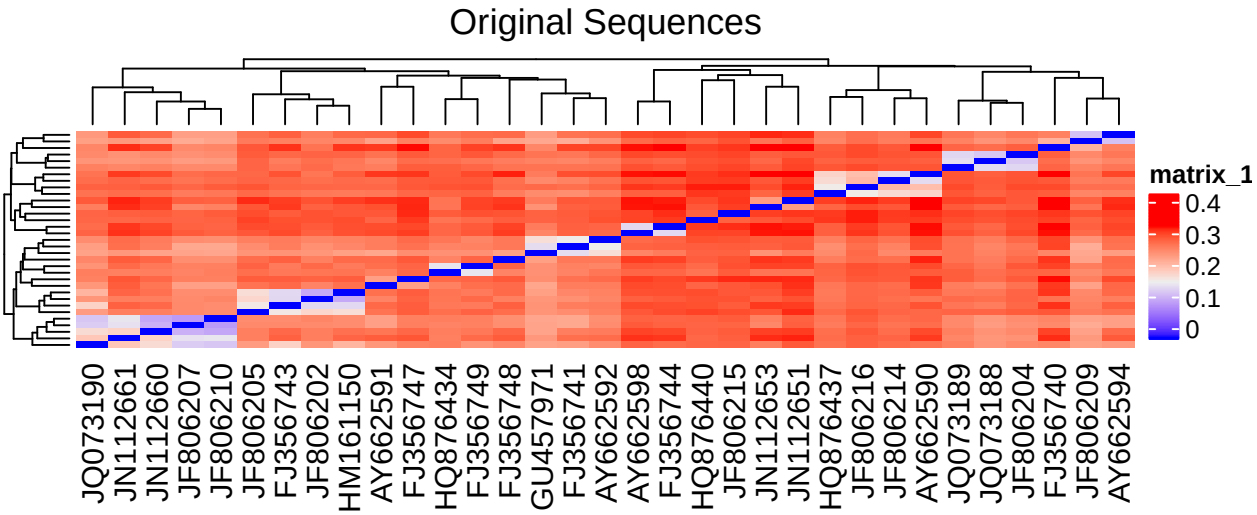
We looked up the source code at GitHub (https://github.com/cran/seqinr/blob/master/R/dist.alignment.R#L27) where it says, that the C distance function is used (line 27). The call looks like this:

```
dist <- .Call("distance", sequences, nbseq, matNumber, seqtype,0, PACKAGE = "seqinr")
```

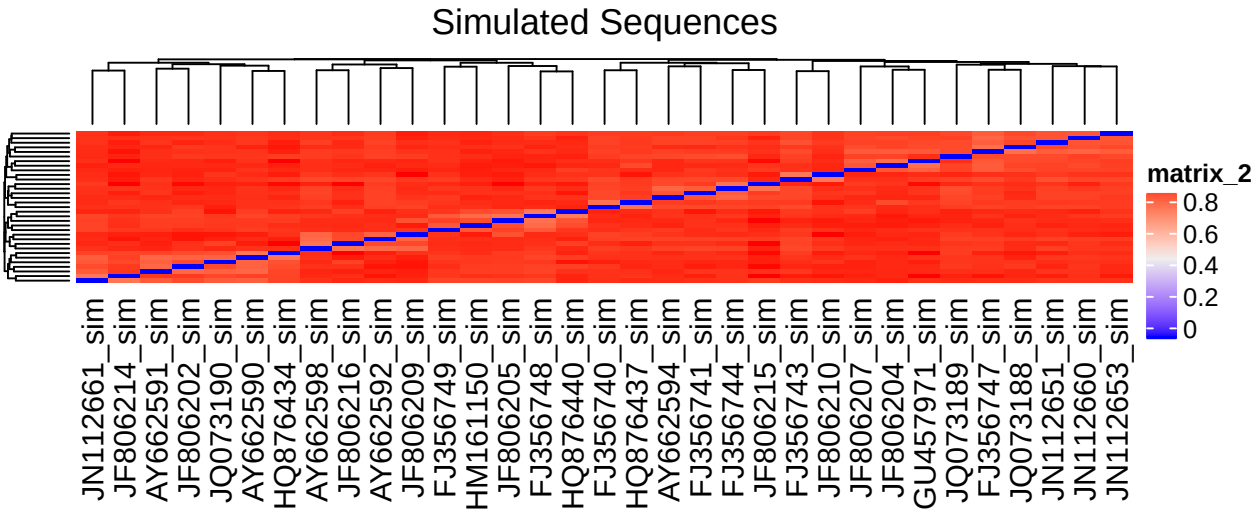
The C code can be found at https://github.com/cran/seqinr/blob/master/src/alignment.c#L256.

Here are the generated Heatmaps visualizing the distance matrices:

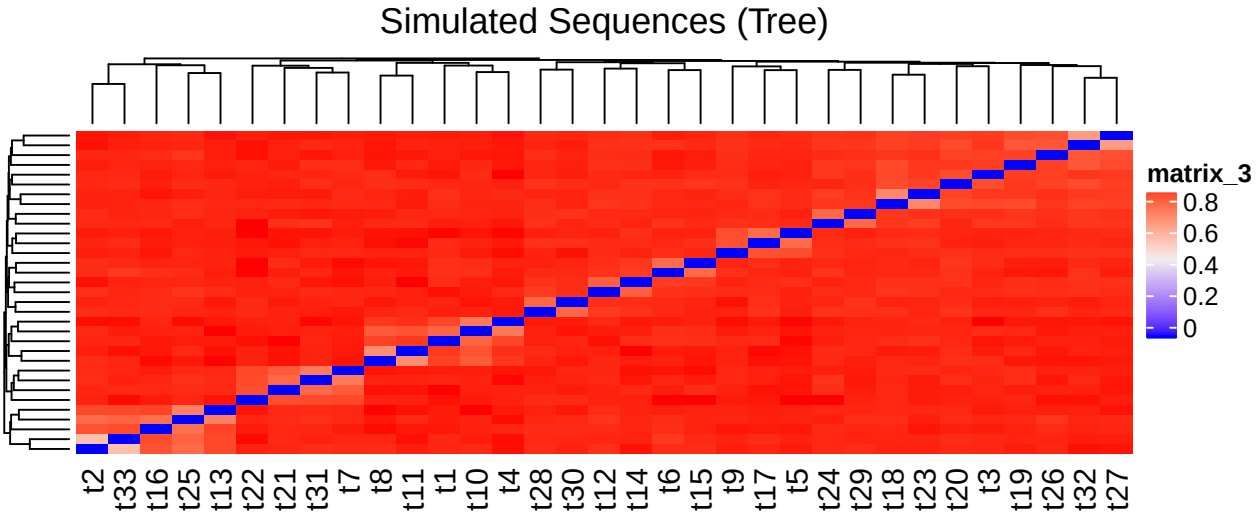
Heatmap for the original sequences.



Heatmap for the simulated sequences.



Heatmap for the simulated sequences (tree).



Small distances are blue with a value of 0.0 while greater distances are red with a value of 1.0. We can see that, of course, the diagonal shows a distance of 0.0 from one sequence to itself. The simulated data, including the simulated tree, has on average a higher distance. Only around the diagonal it can be seen that some sequences have a greater similarity. The original dataset shows that the sequences on average are way more similar.

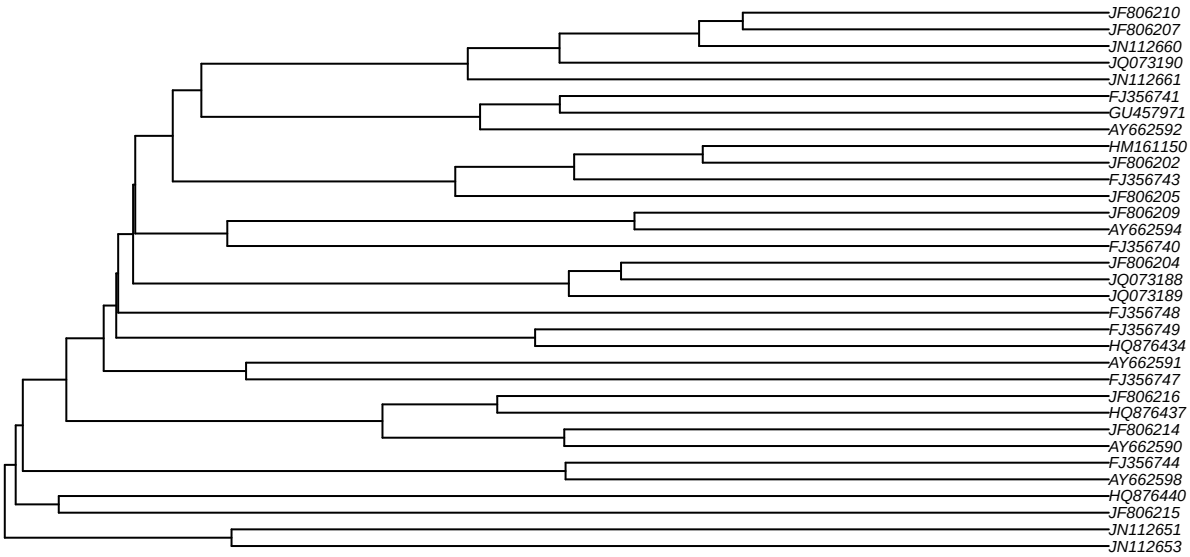
We assume the reason for this is that the original data is indeed not independent, which is the same conslusion we could observe trying fitting the markov chains. The simulated data is independent, therefore we cannot observe that many similarities. The only thing which sticks out is, that on the diagonal, we can observe some sequences which seem to be related.

Question 3

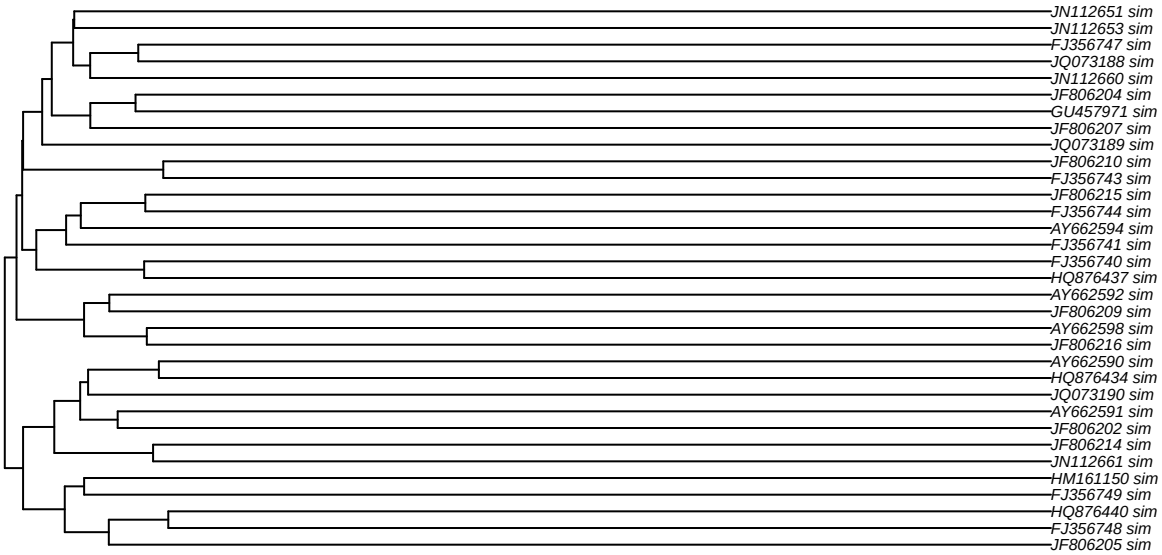
Question 3.1

Construct phylogenetic trees from the three multiple alignments (or distance matrices)

UPGMA tree of the original data

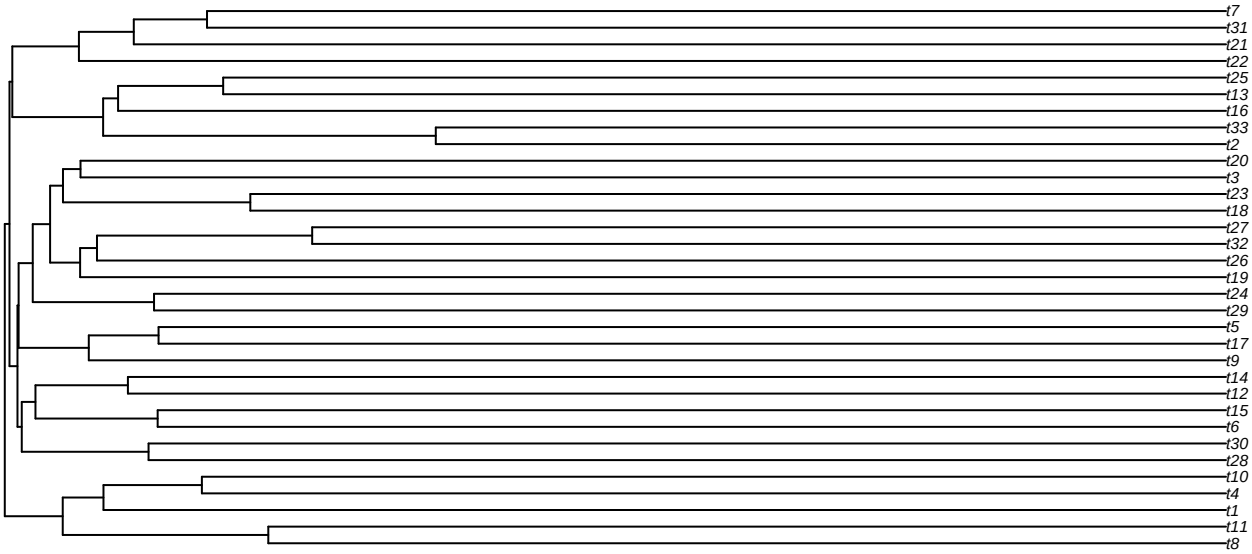


UPGMA tree of the simulated data



UPGMA tree of the simulated data (tree)

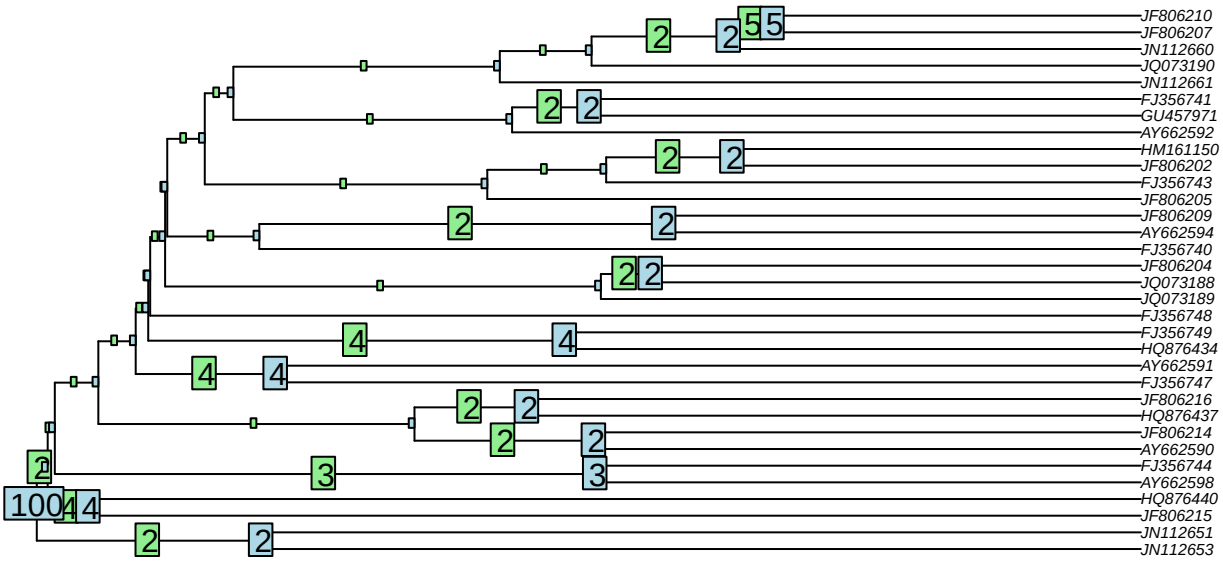
Estimated tree from simulated data according to the tree structure.



Perform a phylogenetic bootstrap analysis

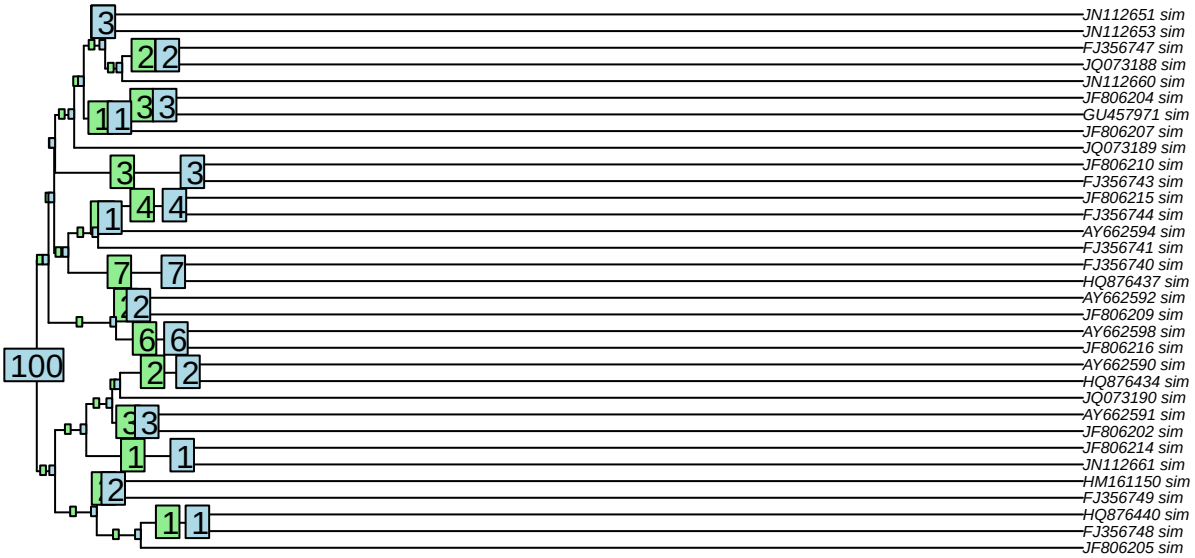
Bootstrap of original sequences

UPGMA tree of the original data with the bootstrap support for the individual clades.



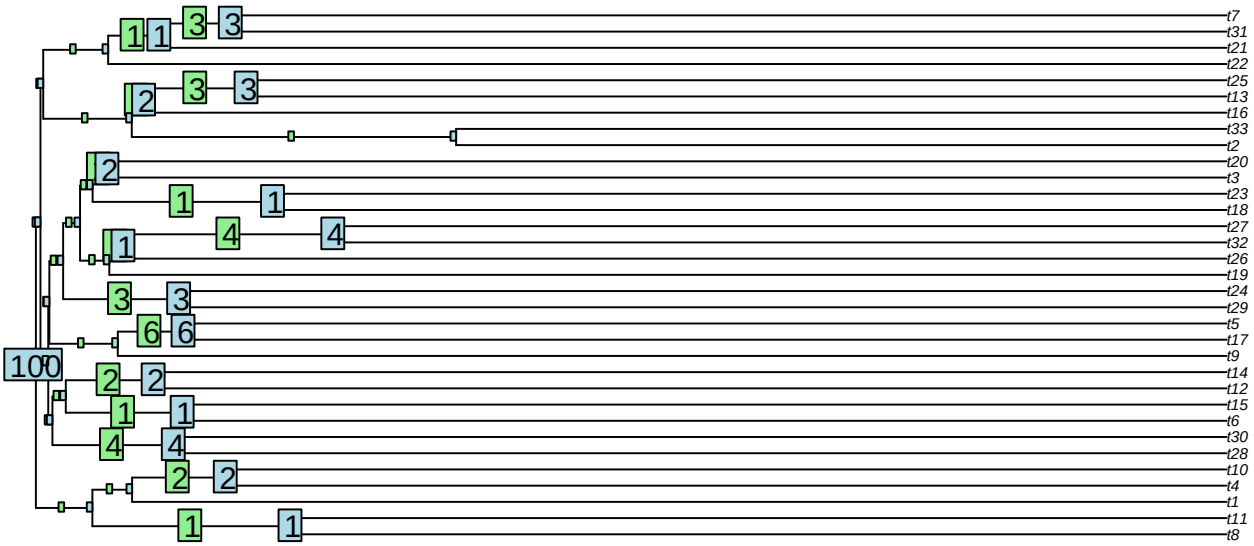
Bootstrap of simulated sequences

UPGMA tree of the simulated data with the bootstrap support for the individual clades.



Bootstrap of simulated sequences from tree

UPGMA tree of the simulated data (from tree) with the bootstrap support for the individual clades.



Question 3.2

Compare your inferred trees and also your simulated one - calculate various indices related to them and distances between the trees

Appendix

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

library(dplyr)
library(tidyr)
library(magrittr)
library(ape)           # This is a general R-package for phylogenetics
                       # and comparative methods
library(seqinr)        # This is an specialized package for
                       # nucleotide sequence management
library(phangorn)
library(knitr)
library(markovchain)  # For fitting, evaluating markov chains (question 2)

# Use this if BiocManager is not installed
#if (!requireNamespace("BiocManager", quietly = TRUE))
#    install.packages("BiocManager")
#library("BiocManager")

# BiocManager packages
#BiocManager::install("ComplexHeatmap", version = "3.8")

library(ComplexHeatmap)
library(circlize)
source("732A51_BioinformaticsHT2018_Lab02_GenBankGetCode.R")
```

```r
# ------------------------------------------------------------------------------
# Question 1.1
# ------------------------------------------------------------------------------

lizards_format_sequences = read.fasta(file = "data/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
  # that are equal to the original ones.
  # NOTE: we use the general distribution for every single sequence
}

# Save as fasta file
write.dna(simulated_lizards, file = "data/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)
# base.freq computes the frequencies of the four DNA bases from a sample of
# sequences.

kableExtra::kable(df_table, booktabs = T, align = c("r", "l", "l"),
      col.names = c("Base", "Original\nfrequency", "Simulated\nfrequency"),
      format = "latex", caption = "Base frequencies of the 33 original and
      generated DNA sequences.", digits = c(NA, 4, 4)) %>%
  kableExtra::kable_styling(latex_options = "hold_position")


# ------------------------------------------------------------------------------
# Question 1.2
# ------------------------------------------------------------------------------

# Simulate phylogenetic tree with 33 tips in phylo format (ape) ----------------
set.seed(1)
tree = ape::rtree(n = 33)
```

```r
# Simulate sequences on this tree using phangorn::simSeq() ---------------------
Q = matrix(c(.1, .8, .05, .05,
             .35, .1, .1, .45,
             .3, .2, .2, .3,
             .6, .1, .25, .05), nrow = 4, byrow = TRUE)
rownames(Q) = c("a", "c", "g", "t")
colnames(Q) = c("a", "c", "g", "t")

Original = p

tree_sequences_sim = phangorn::simSeq(tree, l = 2000, Q = Q, bf = Original)

# Explanation of parameters:
# l = 2000 because average sequence length in given data is ca. 2000
# bf = Original because this is the vector with the original base proportions
# Q = just chosen the matrix from Special Exercise 1 (Question 3)

# Convert to DNAbin
tree_sequences_sim = as.DNAbin(tree_sequences_sim)

# Save simulated sequences as fasta file ---------------------------------------

# Write simulated lizard sequences as fasta file
ape::write.dna(tree_sequences_sim, file ="data/simulated_lizards_tree.fasta",
               format = "fasta", append = F, nbcol = 6, colsep = " ", colw = 10)

# Report base composition ------------------------------------------------------

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

kableExtra::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 trees.") %>%
  kableExtra::kable_styling(latex_options = "hold_position")

# Plot the tree ----------------------------------------------------------------

plot(tree, edge.width = 1, main = "Plot of simulated phylogenetic tree")
# phytools::plotTree(tree) # Alternative

# ------------------------------------------------------------------------------
# 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")
```

```r
sim_tree = ape::read.FASTA(file = "data/simulated_lizards_tree.fasta", type = "DNA")

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

# Individual base composition
kableExtra::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", booktabs = T) %>%
  kableExtra::kable_styling(latex_options = "hold_position")

# GC content
kableExtra::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", booktabs = T) %>%
  kableExtra::kable_styling(latex_options = "hold_position")

# AT content
kableExtra::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", booktabs = T) %>%
  kableExtra::kable_styling(latex_options = "hold_position")


# 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 (default)
    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
```

```r
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) # Add X = 0 to simulated sample
Simulated_Tree = c(Simulated_Tree, X = 0)     # Add X = 0 to simulated sample
Original = Original[c(1:(length(Original)-2), # Let X be last value in original
                      length(Original), length(Original)-1)]

kableExtra::kable(data.frame(Name = c(seqinr::aaa(), "gaps"),
                         Original,
                         Simulated_Sample,
                         Simulated_Tree),
             caption = "Amino Acid Composition (in \\%)", booktabs = T) %>%
  kableExtra::kable_styling(latex_options = "hold_position")


# 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)

kableExtra::kable(t(data.frame(Original = Original["*"],
               Simulated_Sample = Simulated_Sample["*"],
               Simulated_Tree = Simulated_Tree["*"])),
             caption = "Number of Stop Codons", booktabs = T) %>%
  kableExtra::kable_styling(latex_options = "hold_position")


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

# 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
```

```r
# the hypothesis that the sequence is of first order.

cat("Assess the order with Chi-Square test:\n")

cat("Original: p = ",
assessOrder(original_seq, verbose = FALSE)$p.value)

cat("\nSimulated_Sample: p =",
assessOrder(sim_sample_seq, verbose = FALSE)$p.value)

cat("\nSimulated_Tree: p =",
assessOrder(sim_tree_seq, verbose = FALSE)$p.value)

# 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
# -----------------------------------------------------------------------------
# It'd be nice to do the multiple sequence alignment in R code, but due to the
# fact that for the moment it's not working and that it takes a long compilation
# time (>1 minute) we will use Clustal Omega for now:
# (https://www.ebi.ac.uk/Tools/msa/clustalo/)
# In the stack overflow thread Krzysztof suggested the package 'DECIPHER' is
# also recommended.: (http://www2.decipher.codes/Alignment.html)
# The progress of the testing of the multiple sequence alignment in R can be
# found in multi_sequence_alignment.R
# For the heatmapes ths may be used: https://davetang.org/muse/2018/05/15/making-a-heatmap-in-r-with-th

# Read the DNA alignments
original_as_alignment =
  read.alignment("data/aligned_clustalo_lizard_seqs.fasta",
                 format = "fasta")
sim_sample_as_alignment =
```

```
  read.alignment("data/aligned_clustalo_simulated_lizards.fasta",
                 format = "fasta")
sim_tree_as_alignment =
  read.alignment("data/aligned_clustalo_simulated_lizards_tree.fasta",
                 format = "fasta")

# Get the distances (useful in 3.1). The distance function is "unclear", the
# documentation says "specified distance measure". We can also use a different
# one.
dist_original =
  dist.alignment(original_as_alignment, matrix = "identity")
dist_sim_sample =
  dist.alignment(sim_sample_as_alignment, matrix = "identity")
dist_sim_tree =
  dist.alignment(sim_tree_as_alignment, matrix = "identity")

# Get the distance matrixes (for heatmaps)
dist_mat_original = Biostrings::as.matrix(dist_original)
dist_mat_sim_sample = Biostrings::as.matrix(dist_sim_sample)
dist_mat_sim_tree =Biostrings::as.matrix(dist_sim_tree)


# Heatmap examples cen be found here:
# https://bioconductor.org/packages/release/bioc/vignettes/ComplexHeatmap/inst/doc/s2.single_heatmap.ht
# If you don't like Heatmeap, we can also use someting else
Heatmap(dist_mat_original, column_title = "Original Sequences",
        show_row_names = FALSE)


Heatmap(dist_mat_sim_sample, column_title = "Simulated Sequences",
        show_row_names = FALSE)


Heatmap(dist_mat_sim_tree, column_title = "Simulated Sequences (Tree)",
        show_row_names = FALSE)

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

# Answers based on phagron package vignette
# https://cran.r-project.org/web/packages/phangorn/vignettes/Trees.pdf
# Rooted trees

tree_upgma_original = upgma(dist_original)
tree_upgma_sim_sample = upgma(dist_sim_sample)
tree_upgma_sim_tree = upgma(dist_sim_tree)


plot(tree_upgma_original, type = "phylogram",
     main="", cex = 0.5, no.margin = T)
```

```r
plot(tree_upgma_sim_sample, type = "phylogram", cex = 0.5, no.margin = T,
     main="")


plot(tree_upgma_sim_tree, type = "phylogram", cex = 0.5, no.margin = T,
     main="")


# Bootstrapping
# Followed the examples in ?boot.phylo
boot_original = ape::boot.phylo(nj(dist_mat_original), dist_mat_original,
                                nj, trees = T)$trees
clad_original = prop.clades(tree_upgma_original, boot_original, rooted = TRUE)
boot = prop.clades(tree_upgma_original, boot_original)


plot(tree_upgma_original, type = "phylogram",
     main="", cex = 0.5, no.margin = T)
drawSupportOnEdges(boot)
nodelabels(clad_original)


# Bootstrapping
# Followed the examples in ?boot.phylo
boot_sim_sample = ape::boot.phylo(nj(dist_mat_sim_sample), dist_mat_sim_sample,
                                nj, trees = T)$trees
clad_sim_sample = prop.clades(tree_upgma_sim_sample, boot_sim_sample, rooted = T)
boot = prop.clades(tree_upgma_sim_sample, boot_sim_sample)


plot(tree_upgma_sim_sample, type = "phylogram",
     main="", cex = 0.5, no.margin = T)
drawSupportOnEdges(boot)
nodelabels(clad_sim_sample)


# Bootstrapping
# Followed the examples in ?boot.phylo
boot_sim_tree = ape::boot.phylo(nj(dist_mat_sim_tree), dist_mat_sim_tree,
                                nj, trees = T)$trees
clad_sim_tree = prop.clades(tree_upgma_sim_tree, boot_sim_tree, rooted = T)
boot = prop.clades(tree_upgma_sim_tree, boot_sim_tree)



plot(tree_upgma_sim_tree, type = "phylogram",
     main="", cex = 0.5, no.margin = T)
drawSupportOnEdges(boot)
nodelabels(clad_sim_tree)

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

```
###from TotalCopheneticIndex package :
library(TotalCopheneticIndex)

##tci() - it calculates the total cophenetic index for any tree
##        The Total Cophenetic Index is a measure of tree balance.
total_index_original = tci(tree_upgma_original)
total_index_sample = tci(tree_upgma_sim_sample)
total_index_tree = tci(tree_upgma_sim_tree)

##tci.context() - Calculate the range of values that the Total Cophenetic Index
#can take, and expected values under the Yule and Uniform models of evolution
range_index_original = tci.context(tree_upgma_original)
range_index_sample = tci.context(tree_upgma_sim_sample)
range_index_tree = tci.context(tree_upgma_sim_tree)


###distance between trees
library(phangorn)
dist1 = treedist(tree_upgma_original,tree_upgma_sim_sample, check.labels = F)
dist2 = treedist(tree_upgma_original,tree_upgma_sim_tree, check.labels = F)
dist3 = treedist(tree_upgma_sim_sample,tree_upgma_sim_tree, check.labels = F)

#RF.dist - get the distance that only depends on the topology of the trees.
RF.dist1 = RF.dist(tree_upgma_original,tree_upgma_sim_sample, check.labels = F)
RF.dist2 = RF.dist(tree_upgma_original,tree_upgma_sim_tree, check.labels = F)
RF.dist3 = RF.dist(tree_upgma_sim_sample,tree_upgma_sim_tree, check.labels = F)
```