Bioinformatics - Computer Lab 2

Group 7: Lennart Schilling (lensc874), Thijs Quast (thiqu264), Mariano Maquieira Mariani (marma330)

27 November 2018

Question 1

At first, the dataset of the RAG1 gene sequences from 33 lizard species were downloaded from GenBank and saved in a fasta file using the provided R script *732A51 BioinformaticsHT2018 Lab02 GenBankGetCode.R.*. The code can be found in Appendix 1 (Data Import of original dataset).

Question 1.1

The saved fasta-file has to be read in R so that we can work with that. After analysing the sequences, it becomes clear that there can be found many whitespaces (“”). Since the artificial sequences should be simulated so that each nucleotide is to be independently and randomly drawn from the distribution given by the base composition in the true lizard sequences, the whitespaces have to be removed. Otherwise the artificial sequences are built on a probability distribution where the sum of all probabilities would not equal 1. The R code for the reading and preparation process can be found in Appendix 1.1 (Reading and preparing original data).

After preparing the data, the artificial dataset is built by considering that it contains 33 sequences (each length of the sequences is the same as in the lizard dataset) so that for each real sequence an artificial one is created. As mentioned, the simulation of the artificial sequences is based on the distribution given by the base composition of the original dataset.

The artificial dataset is submitted as the fasta file *artificial_dataset_1_1.fasta*. The written function for all these processes automatically prints the base composition in the simulated data compared to the base composition in the original data. An extract from the output can be seen here:

```
get_artificial_sequence_dataset(lizards_sequences)
```

```
## [1] "comparison of base compositions between original and artificial datasets (values rounded):"
##   name_original name_artificial a_original a_artificial c_original
## 1 "JF806202"    "1"             "0.29"     "0.27"       "0.2"
## 2 "HM161150"    "2"             "0.31"     "0.31"       "0.21"
## 3 "FJ356743"    "3"             "0.31"     "0.31"       "0.21"
## 4 "JF806205"    "4"             "0.28"     "0.28"       "0.21"
## 5 "JQ073190"    "5"             "0.31"     "0.33"       "0.2"
##   c_artificial g_original g_artificial t_original t_artificial
## 1 "0.21"       "0.24"     "0.24"       "0.26"     "0.27"
## 2 "0.23"       "0.23"     "0.23"       "0.24"     "0.23"
## 3 "0.22"       "0.23"     "0.24"       "0.24"     "0.23"
## 4 "0.2"        "0.24"     "0.24"       "0.26"     "0.28"
## 5 "0.19"       "0.24"     "0.23"       "0.26"     "0.25"
```

It becomes clear that the base compositions are very similar. The entire code for the function can be seen in Appendix 1.1 (Function code).

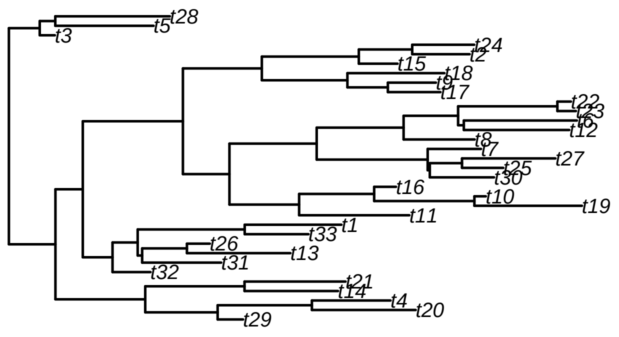
Question 1.2

In this part of the exercise do we use the prepared data from part 1, in Appendix 1 code can be found in (Data Import of original dataset).

We used the function *rtree* to create a tree object of the type phylo and the length of the original sequences.

```
tree <- rtree(n = length(lizards_sequences))
```

Here you can find the plot of the tree.



After the simulation of the phylogenetic tree, we had to simulate the sequence.

For this, we the had several things to do. 1. We simulated a transition rate matrix (Q-Matrix). In this case we choose one by yourself.

```
##      a    c    t    g
## a 0.25 0.25 0.25 0.25
## c 0.25 0.25 0.25 0.25
## t 0.25 0.25 0.25 0.25
## g 0.25 0.25 0.25 0.25
```

2. We had to choose the length of the sequence. To make it comparable with the original lizards dataset, we decided to create

```
lengths <- c()
for (i in 1:33){
lengths <- c(lengths, length(lizards_sequences[[i]]))
}
```

Now we can use the simulate the sequences by using the function *phangorn::simSeq()*.

```
sequences_artificial <- list()
for (j in 1:33){
sequences_artificial[j] <- simSeq(tree, l = lengths[j], Q=transition_matrix , type = "DNA")
}
```

Since in sequences are filled with integers from 1 to 4, do we have to replace the numbers by the letters a,b,c,d.

1 = a

2 = b

3 = c

4 = d

The code for this can be found in Appendix 1.2

The second simulate a artificial DNA sequence dataset do we save as *"artificial_dataset_1_2.fasta"*.

```
ape::write.dna(sequences_artificial, file ="artificial_dataset_1_2.fasta", format = "fasta", colsep = "
```

Question 2

Question 2.1

```
lizards_sequences = read.fasta("lizard_seqs.fasta")
original_dataset <- lizards_sequences
artificial_sequences_1 <- read.fasta("artificial_dataset_1_1.fasta")
artificial_sequences_2 <- read.fasta("artificial_dataset_1_2.fasta")
original_base_compositions <- list()
artificial_1_base_compositions <- list()
artificial_2_base_compositions <- list()
for (i in 1:length(original_dataset)) {
    # getting base compositions for each original sequence
    original_base_compositions[[i]] =
      seqinr::count(original_dataset[[i]],1)
}

for (i in 1:length(artificial_sequences_1)) {
    # getting base compositions for each original sequence
    artificial_1_base_compositions[[i]] =
      seqinr::count(artificial_sequences_1[[i]],1)
}

for (i in 1:length(artificial_sequences_2)) {
    # getting base compositions for each original sequence
    artificial_2_base_compositions[[i]] =
      seqinr::count(artificial_sequences_2[[i]],1)
}
```

```
Reduce('+', original_base_compositions)
```

```
##
##     a     c     g     t
## 20414 13422 15089 16474
```

```
sum(Reduce('+', original_base_compositions))
```

```
## [1] 65399
```

```
Reduce('+', original_base_compositions)/sum(Reduce('+', original_base_compositions))
```

```
##
##         a         c         g         t
## 0.3121454 0.2052325 0.2307222 0.2518999
```

```
Reduce('+', artificial_1_base_compositions)
```

```
##
##     a     c     g     t
## 20405 13503 15115 16412
```

```
sum(Reduce('+', artificial_1_base_compositions))
```

```
## [1] 65435
```

```
Reduce('+', artificial_1_base_compositions)/sum(Reduce('+', artificial_1_base_compositions))
```

```
##
##         a         c         g         t
## 0.3118362 0.2063575 0.2309926 0.2508138
```

```
Reduce('+', artificial_2_base_compositions)
```

```
##
##     a     c     g     t
## 16327 16212 16346 16550
```

```
sum(Reduce('+', artificial_2_base_compositions))
```

```
## [1] 65435
```

```
Reduce('+', artificial_2_base_compositions)/sum(Reduce('+', artificial_2_base_compositions))
```

```
##
##         a         c         g         t
## 0.2495148 0.2477573 0.2498052 0.2529227
```

The original dataset and the first artificially created dataset are rather similar in their distributions for A, C, T and G's. However, the second artificially created dataset has a slightly different distribution. This final dataset has almost uniform distribution for A, C, T and G's, they all occur with an average frequency of approximately 25%.

```
library(rDNAse)
original_compositions <- list()
  for (i in 1:length(lizards_sequences)) {
  string1 <- paste(lizards_sequences[[i]], collapse = "")
  string1 <- toupper(string1)
  original_compositions[[i]] <- kmer(string1)
}
```

```
artificial_compositions_1 <- list()
  for (i in 1:length(artificial_sequences_1)) {
  string1 <- paste(artificial_sequences_1[[i]], collapse = "")
  string1 <- toupper(string1)
  artificial_compositions_1[[i]] <- kmer(string1)
}
```

```
artificial_compositions_2 <- list()
  for (i in 1:length(artificial_sequences_2)) {
  string1 <- paste(artificial_sequences_2[[i]], collapse = "")
  string1 <- toupper(string1)
  artificial_compositions_2[[i]] <- kmer(string1)
}
```

```r
Reduce('+', original_compositions)
```

```
##      TG      GA      AA      AG      AT      TT      CA      CT      TC
##    6377    5456    5292    5060    4726    4263    4066    3880    3339
##      GC      CC      AC      GG      GT (Other)    NA's
##    3183    3040    2912    2791    2592    2905   10971
```

```r
Reduce('+', artificial_compositions_1)
```

```
##   AA   AC   AG   AT   CA   CC   CG   CT   GA   GC   GG   GT   TA   TC   TG
## 6498 4145 4728 5022 4129 2777 3112 3476 4688 3146 3483 3792 5081 3425 3789
##   TT
## 4111
```

```r
Reduce('+', artificial_compositions_2)
```

```
##   AA   AC   AG   AT   CA   CC   CG   CT   GA   GC   GG   GT   TA   TC   TG
## 4104 4026 4140 4047 4109 4040 3983 4074 4011 4096 4075 4156 4094 4045 4136
##   TT
## 4266
```

GC content is the largest for the second artificially created dataset. CG content is largest for the second artificially created dataset. AT content is largest in the original dataset.

```r
# Protein sequences
protein_original <- read.fasta("lizard_protein.fasta")
protein_artificial_1 <- read.fasta("artificial_1_protein.fasta")
protein_artificial_2 <- read.fasta("artificial_2_protein.fasta")
```

```r
library(protr)
original_aac <- list()
for (i in 1:length(protein_original)) {
string1 <- paste(protein_original[[i]], collapse = "")
string1 <- toupper(string1)
string1 <- gsub(pattern = "[*]", replacement = "", x = string1)
string1 <- gsub(pattern = "B", replacement = "", x = string1)
string1 <- gsub(pattern = "J", replacement = "", x = string1)
string1 <- gsub(pattern = "O", replacement = "", x = string1)
string1 <- gsub(pattern = "U", replacement = "", x = string1)
string1 <- gsub(pattern = "X", replacement = "", x = string1)
string1 <- gsub(pattern = "Z", replacement = "", x = string1)
original_aac[[i]] <- extractAAC(string1)
}
```

```r
artificial_1_aac <- list()
for (i in 1:length(protein_artificial_1)) {
string1 <- paste(protein_artificial_1[[i]], collapse = "")
string1 <- toupper(string1)
string1 <- gsub(pattern = "[*]", replacement = "", x = string1)
string1 <- gsub(pattern = "B", replacement = "", x = string1)
string1 <- gsub(pattern = "J", replacement = "", x = string1)
string1 <- gsub(pattern = "O", replacement = "", x = string1)
string1 <- gsub(pattern = "U", replacement = "", x = string1)
string1 <- gsub(pattern = "X", replacement = "", x = string1)
string1 <- gsub(pattern = "Z", replacement = "", x = string1)
artificial_1_aac[[i]] <- extractAAC(string1)
}
```

```
artificial_2_aac <- list()
for (i in 1:length(protein_artificial_2)) {
string1 <- paste(protein_artificial_2[[i]], collapse = "")
string1 <- toupper(string1)
string1 <- gsub(pattern = "[*]", replacement = "", x = string1)
string1 <- gsub(pattern = "B", replacement = "", x = string1)
string1 <- gsub(pattern = "J", replacement = "", x = string1)
string1 <- gsub(pattern = "O", replacement = "", x = string1)
string1 <- gsub(pattern = "U", replacement = "", x = string1)
string1 <- gsub(pattern = "X", replacement = "", x = string1)
string1 <- gsub(pattern = "Z", replacement = "", x = string1)
artificial_2_aac[[i]] <- extractAAC(string1)
}
```

```
Reduce('+', original_aac)/length(original_aac)
```

```
##          A          R          N          D          C          E
## 0.04504567 0.06892454 0.03459927 0.03899492 0.04615282 0.06251367
##          Q          G          H          I          L          K
## 0.04665052 0.05212339 0.03881790 0.03970175 0.09512420 0.06888196
##          M          F          P          S          T          W
## 0.02177691 0.04072228 0.06041408 0.09363222 0.05396491 0.02155234
##          Y          V
## 0.02414733 0.04625931
```

```
Reduce('+', artificial_1_aac)/length(artificial_1_aac)
```

```
##          A          R          N          D          C          E
## 0.05057534 0.08796184 0.04476749 0.03495967 0.03066037 0.04049515
##          Q          G          H          I          L          K
## 0.04055862 0.05711907 0.02888776 0.06421406 0.09216594 0.05237905
##          M          F          P          S          T          W
## 0.01695355 0.03323242 0.04400956 0.09865235 0.06662918 0.01463302
##          Y          V
## 0.03807384 0.06307173
```

```
Reduce('+', artificial_2_aac)/length(artificial_2_aac)
```

```
##          A          R          N          D          C          E
## 0.06681774 0.09424995 0.03355575 0.03320151 0.03318094 0.03266863
##          Q          G          H          I          L          K
## 0.03317184 0.06386547 0.03403190 0.04785987 0.09575508 0.03288863
##          M          F          P          S          T          W
## 0.01713940 0.03254250 0.06717763 0.10010313 0.06718395 0.01835640
##          Y          V
## 0.03379739 0.06245228
```

After removing some unwanted letters and characters, the observed amino acids remain for the obtained protein sequences. Distribution of the amino acids among the three databases of obtained protein sequences is rather similar for all three protein databases.

```
library(seqinr)
library(stringr)

# reading original_dataset from fasta file
lizards_sequences = read.fasta("lizard_seqs.fasta")
```

```r
# preparing data in fasta file (dna sequences include emtpy spaces which will be removed)
for (i in 1:length(lizards_sequences)) {
  lizards_sequences[[i]] = lizards_sequences[[i]][lizards_sequences[[i]] != " "]
}
taa_count <- c()
tag_count <- c()
tga_count <- c()

for (i in 1:33){
string <- lizards_sequences[[i]]
string <- paste(lizards_sequences[[i]], collapse = "")
taa_count[i] <-str_count(string, pattern = "taa")
tag_count[i] <- str_count(string, pattern = "tag")
tga_count[i] <- str_count(string, pattern = "tga")
}

names_sequences <- names(lizards_sequences)
df_original <- as.data.frame(cbind(names_sequences, taa_count, tag_count, tga_count,
                                   total_count_1 = taa_count + tag_count + tga_count))

artificial_sequences_1 <- read.fasta("artificial_dataset_1_1.fasta")
taa_a1 <- c()
tag_a1 <- c()
tga_a1 <- c()
for (i in 1:33){
  string <- artificial_sequences_1[[i]]
  string <- paste(artificial_sequences_1[[i]], collapse = "")
  taa_a1[i] <-str_count(string, pattern = "taa")
  tag_a1[i] <- str_count(string, pattern = "tag")
  tga_a1[i] <- str_count(string, pattern = "tga")
}

names_a1 <- names(artificial_sequences_1)

df_a1 <- as.data.frame(cbind(names_a1, taa_a1, tag_a1, tga_a1, total_count_2 =
                               taa_a1 + tag_a1 + tga_a1))

artificial_sequences_2 <- read.fasta("artificial_dataset_1_2.fasta")
taa_a2 <- c()
tag_a2 <- c()
tga_a2 <- c()

for (i in 1:33){
  string <- artificial_sequences_2[[i]]
  string <- paste(artificial_sequences_2[[i]], collapse = "")
  taa_a2[i] <-str_count(string, pattern = "taa")
  tag_a2[i] <- str_count(string, pattern = "tag")
  tga_a2[i] <- str_count(string, pattern = "tga")
}

names_a2 <- names(artificial_sequences_1)

df_a2 <- as.data.frame(cbind(names_a2, taa_a2, tag_a2, tga_a2, total_count_3 =
                                taa_a2 + tag_a2 + tga_a2))

df_all <- as.data.frame(cbind(df_a1, df_a2))
df_all
```

```
##    names_a1 taa_a1 tag_a1 tga_a1 total_count_2 names_a2 taa_a2 tag_a2
## 1         1     22     22     22            66        1     23     16
## 2         2     60     51     35           146        2     42     37
## 3         3     56     55     51           162        3     38     54
## 4         4     28     17     17            62        4     18     17
## 5         5     49     29     27           105        5     19     26
## 6         6     23     28     21            72        6     18      8
## 7         7     67     32     54           153        7     43     51
## 8         8     26     20     12            58        8     16     10
## 9         9     24     23     15            62        9     21     15
## 10       10     64     64     50           178       10     46     50
## 11       11     62     59     54           175       11     39     43
## 12       12     71     65     61           197       12     43     56
## 13       13     69     35     57           161       13     50     37
## 14       14     68     57     61           186       14     43     39
## 15       15     68     54     62           184       15     47     48
## 16       16     30     14     15            59       16     14     13
## 17       17     26     18     22            66       17     14     15
## 18       18     70     38     62           170       18     45     38
## 19       19     74     54     58           186       19     38     46
## 20       20     23     18     19            60       20     20     20
## 21       21     43     24     26            93       21     33     25
## 22       22     70     50     47           167       22     40     48
## 23       23     30     24     34            88       23     22     25
## 24       24     21     10     12            43       24     19     10
## 25       25     75     34     59           168       25     46     47
## 26       26     68     45     43           156       26     35     31
## 27       27     37     17     14            68       27     16     19
## 28       28     71     52     50           173       28     40     63
## 29       29     66     49     54           169       29     43     55
## 30       30     29     22     23            74       30     15     19
## 31       31     84     54     44           182       31     42     45
## 32       32     29     17     15            61       32     14     24
## 33       33     18     23     22            63       33     11     13
##    tga_a2 total_count_3
## 1      10            49
## 2      51           130
## 3      45           137
## 4      18            53
## 5      26            71
## 6      27            53
## 7      45           139
## 8      24            50
## 9      17            53
## 10     50           146
## 11     45           127
## 12     45           144
## 13     43           130
## 14     44           126
## 15     40          135
## 16     15           42
## 17     16           45
## 18     44          127
## 19     41          125
## 20     15           55
## 21     21           79
## 22     52          140
## 23     25           72
## 24     13           42
## 25     53          146
## 26     62          128
## 27     15           50
## 28     42          145
## 29     38          136
## 30     17           51
## 31     42          129
## 32     16           54
## 33     18           42
```

Interpreting stop codons as either “taa”, “tag” or “tga” results in many stop codons for each sequence. In the original dataset this is highly unlikely, as a natural translation starts at a start codon and then continues until it reaches a stop codon. Or if it does not reach a stop codon at all.

Question 2.2

```r
library(markovchain)
mcFitMle_original <- markovchainFit(lizards_sequences, method = "mle")
mcFitMle_original
```

```
## $estimate
## MLE Fit
##  A  8 - dimensional discrete Markov Chain defined by the following states:
##  a, c, g, m, r, s, t, y
##  The transition matrix  (by rows)  is defined as follows:
##           a         c          g            m            r            s
## a 0.3377604 0.1730948 0.27493261 4.900760e-05 0.0002450380 0.000000e+00
## c 0.3793901 0.2477071 0.05010812 0.000000e+00 0.0003728283 0.000000e+00
## g 0.3934372 0.2029168 0.19323832 6.629102e-05 0.0003314551 0.000000e+00
## m 0.0000000 0.0000000 0.66666667 0.000000e+00 0.0000000000 0.000000e+00
## r 0.4117647 0.1764706 0.11764706 0.000000e+00 0.0000000000 0.000000e+00
## s 0.0000000 1.0000000 0.00000000 0.000000e+00 0.0000000000 0.000000e+00
## t 0.1508047 0.2115396 0.35718190 6.073489e-05 0.0001214698 6.073489e-05
## y 0.3333333 0.2000000 0.13333333 0.000000e+00 0.0000000000 0.000000e+00
##           t            y
## a 0.2136731 0.0002450380
## c 0.3222728 0.0001491313
## g 0.2096122 0.0003977461
## m 0.3333333 0.0000000000
## r 0.2941176 0.0000000000
## s 0.0000000 0.0000000000
## t 0.2801093 0.0001214698
## y 0.3333333 0.0000000000
##
```

```
##
## $standardError
##             a           c           g            m            r
## a 0.004068516 0.002912552 0.003670666 4.900760e-05 1.095843e-04
## c 0.005318784 0.004297725 0.001932963 0.000000e+00 1.667339e-04
## g 0.005106990 0.003667637 0.003579101 6.629102e-05 1.482312e-04
## m 0.000000000 0.000000000 0.471404521 0.000000e+00 0.000000e+00
## r 0.155632430 0.101885342 0.083189033 0.000000e+00 0.000000e+00
## s 0.000000000 1.000000000 0.000000000 0.000000e+00 0.000000e+00
## t 0.003026402 0.003584388 0.004657618 6.073489e-05 8.589211e-05
## y 0.149071198 0.115470054 0.094280904 0.000000e+00 0.000000e+00
##              s           t            y
## a 0.000000e+00 0.003235986 1.095843e-04
## c 0.000000e+00 0.004902089 1.054518e-04
## g 0.000000e+00 0.003727654 1.623792e-04
## m 0.000000e+00 0.333333333 0.000000e+00
## r 0.000000e+00 0.131533410 0.000000e+00
## s 0.000000e+00 0.000000000 0.000000e+00
## t 6.073489e-05 0.004124610 8.589211e-05
## y 0.000000e+00 0.149071198 0.000000e+00
##
## $confidenceLevel
## [1] 0.95
##
## $lowerEndpointMatrix
##            a           c          g m            r s          t
## a 0.33106824 0.168304107 0.26889491 0 6.478782e-05 0 0.20835040
## c 0.37064143 0.240637977 0.04692868 0 9.857546e-05 0 0.31420954
## g 0.38503694 0.196884079 0.18735122 0 8.763643e-05 0 0.20348075
## m 0.00000000 0.000000000 0.00000000 0 0.000000e+00 0 0.00000000
## r 0.15577214 0.008884115 0.00000000 0 0.000000e+00 0 0.07776444
## s 0.00000000 0.000000000 0.00000000 0 0.000000e+00 0 0.00000000
## t 0.14582675 0.205643836 0.34952080 0 0.000000e+00 0 0.27332494
## y 0.08813303 0.010068663 0.00000000 0 0.000000e+00 0 0.08813303
##              y
## a 6.478782e-05
## c 0.000000e+00
## g 1.306561e-04
## m 0.000000e+00
## r 0.000000e+00
## s 0.000000e+00
## t 0.000000e+00
## y 0.000000e+00
##
## $upperEndpointMatrix
##           a         c          g            m            r            s
## a 0.3444525 0.1778856 0.28097032 0.0001296179 0.0004252881 0.0000000000
## c 0.3881387 0.2547762 0.05328756 0.0000000000 0.0006470811 0.0000000000
## g 0.4018374 0.2089495 0.19912541 0.0001753300 0.0005752738 0.0000000000
## m 0.0000000 0.0000000 1.00000000 0.0000000000 0.0000000000 0.0000000000
## r 0.6677573 0.3440571 0.25448084 0.0000000000 0.0000000000 0.0000000000
## s 0.0000000 1.0000000 0.00000000 0.0000000000 0.0000000000 0.0000000000
## t 0.1557827 0.2174354 0.36484300 0.0001606349 0.0002627497 0.0001606349
## y 0.5785336 0.3899313 0.28841162 0.0000000000 0.0000000000 0.0000000000
##           t            y
## a 0.2189958 0.0004252881
## c 0.3303360 0.0003225840
## g 0.2157436 0.0006648361
## m 0.8816179 0.0000000000
## r 0.5104709 0.0000000000
## s 0.0000000 0.0000000000
## t 0.2868937 0.0002627497
## y 0.5785336 0.0000000000
```

```
mcFitMle_a1 <- markovchainFit(artificial_sequences_1, method = "mle")
mcFitMle_a1
```

```
## $estimate
## MLE Fit
##  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 0.3186387 0.2032560 0.2318443 0.2462610
## c 0.3059878 0.2057952 0.2306210 0.2575960
## g 0.3102786 0.2082203 0.2305249 0.2509762
## t 0.3097038 0.2087651 0.2309521 0.2505791
##
##
## $standardError
##             a           c           g           t
## a 0.003952835 0.003157048 0.003371767 0.003475019
## c 0.004761916 0.003905236 0.004134082 0.004369172
## g 0.004531668 0.003712305 0.003906079 0.004075665
## t 0.004344821 0.003567201 0.003751972 0.003908148
##
## $confidenceLevel
## [1] 0.95
##
## $lowerEndpointMatrix
##           a         c         g         t
## a 0.3121369 0.1980631 0.2262982 0.2405451
## c 0.2981552 0.1993716 0.2238211 0.2504093
## g 0.3028247 0.2021141 0.2240999 0.2442724
## t 0.3025572 0.2028976 0.2247806 0.2441507
##
## $upperEndpointMatrix
##           a         c         g         t
## a 0.3251406 0.2084489 0.2373903 0.2519769
## c 0.3138205 0.2122187 0.2374210 0.2647826
## g 0.3177326 0.2143265 0.2369498 0.2576801
## t 0.3168504 0.2146326 0.2371235 0.2570074
```

```
mcFitMle_a2 <- markovchainFit(artificial_sequences_2, method = "mle")
mcFitMle_a2
```

```
## $estimate
## MLE Fit
##  A  4 - dimensional discrete Markov Chain defined by the following states:
```

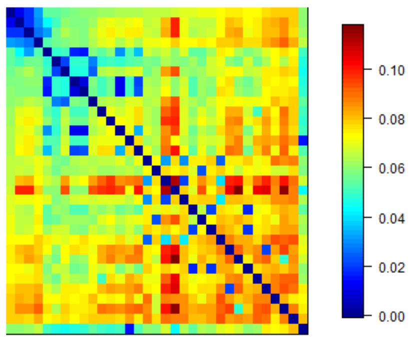
```
##  a, c, g, t
##  The transition matrix  (by rows)  is defined as follows:
##           a         c         g         t
## a 0.2515168 0.2467365 0.2537231 0.2480235
## c 0.2535481 0.2492904 0.2457732 0.2513884
## g 0.2455013 0.2507039 0.2494185 0.2543763
## t 0.2475062 0.2445439 0.2500453 0.2579046
##
##
## $standardError
##             a           c           g           t
## a 0.003926118 0.003888630 0.003943300 0.003898758
## c 0.003955417 0.003922066 0.003894299 0.003938535
## g 0.003876390 0.003917248 0.003907193 0.003945835
## t 0.003868229 0.003845010 0.003888020 0.003948650
##
## $confidenceLevel
## [1] 0.95
##
## $lowerEndpointMatrix
##           a         c         g         t
## a 0.2450589 0.2403403 0.2472370 0.2416106
## c 0.2470420 0.2428392 0.2393676 0.2449101
## g 0.2391252 0.2442606 0.2429918 0.2478860
## t 0.2411435 0.2382194 0.2436501 0.2514096
##
## $upperEndpointMatrix
##           a         c         g         t
## a 0.2579747 0.2531328 0.2602093 0.2544364
## c 0.2600542 0.2557416 0.2521787 0.2578667
## g 0.2518774 0.2571472 0.2558453 0.2608666
## t 0.2538689 0.2508683 0.2564406 0.2643996
```

We fitted a first order markov model on all sequences. Our assumption in our simulated datasets is that in the sequence the occurence of a nucleotide does not depend on the rest of the sequence. This violates the limited horizon: which is that the probability of being in a state at time t depends only on the state at time t minus 1. We used sample {base} function, which obviously samples without taking into account past states.

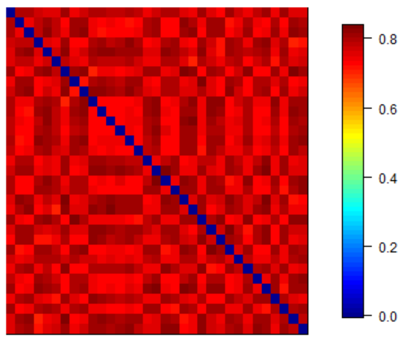
Question 2.3

To allign the sequences for each dataset (the original dataset *lizards_sequences*, the first artificial dataset *artificial_dataset_1_1* and the second artificial dataset *artificial_dataset_1_2*), the *plsgenomics* package was used. The *.fasta*-files for the datasets were transformed to a *DNAStringSet* - class within R. The uncorrected distance matrices created represent the hamming distance between each of the sequences in each dataset. The results of these distance matrices are plotted as heatmaps (using *plsgenomics* package) :

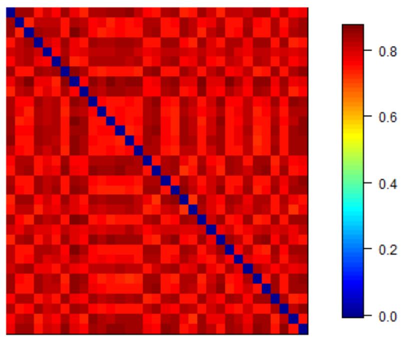
lizards_sequences



artificial_dataset_1_1



artificial_dataset_1_2



We see that for the original dataset, the allignment results are much better than for the artificial datasets. Based on the point that the artificial datasets were created by sampling randomly, the greater distances between the sequences compared to the distances within the original dataset make sense.

The R code for this Question 2.3 can be found in Appendix 2.3.

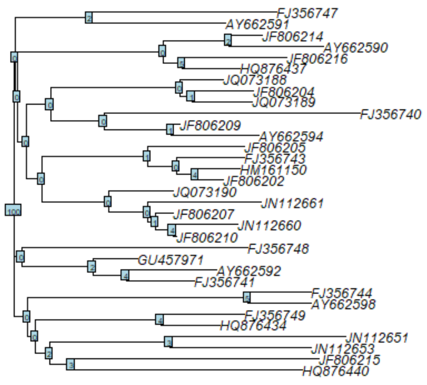
Question 3

Question 3.1

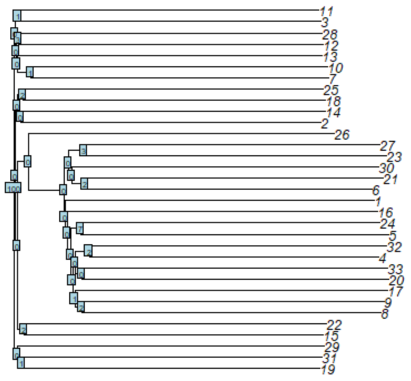
Using the created distance matrix for each dataset (the original dataset *lizards_sequences*, the first artificial dataset *artificial_dataset_1_1* and the second artificial dataset *artificial_dataset_1_2*) with the aligned sequences, phylotrees were created. On top of that, a phylogenetic bootstrap analysis was performed. As a result, the bootstrap supports for the individual clades were integrated into the phylotrees.

```
## detected function mkl_set_num_threads
```

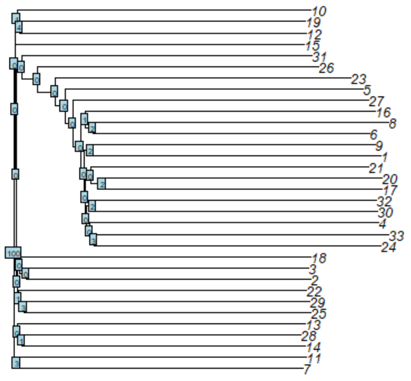
lizards_sequences



artificial_dataset_1_1



artificial_dataset_1_2



The R code for the creation of the phylotrees and the bootstrap analysis can be found in Appendix 3.1.

Question 3.2

Different general characteristics can be comprared between phylogenetic trees, e.g.:

- number of tips
- different tips
- number of nodes

On top of that, different quantitative distances can be calculated, e.g.:

- symmetric difference
- branch score

The distances can be only calculated if the tips are named equally. Since the artificial datasets (*artificial_dataset_1_1* and *artificial_dataset_1_2*) are not named as the original dataset (*lizard_sequences*), the distance measurements could be only processed for the comparison between the artficial datasets.

```
## => Comparing phylotree1 with phylotree2.
## Both trees have the same number of tips: 33.
## Tips in phylotree1 not in phylotree2 : JF806202, HM161150, FJ356743, JF806205, JQ073190, GU457971, F
## Tips in phylotree2 not in phylotree1 : 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18
## Both trees have the same number of nodes: 31.
## Both trees are unrooted.
## Both trees are not ultrametric.

## => Comparing phylotree1 with phylotree2.
## Both trees have the same number of tips: 33.
## Tips in phylotree1 not in phylotree2 : JF806202, HM161150, FJ356743, JF806205, JQ073190, GU457971, F
## Tips in phylotree2 not in phylotree1 : 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18
## Both trees have the same number of nodes: 31.
## Both trees are unrooted.
## Both trees are not ultrametric.

##      symmetric.difference   branch.score.difference
##               56.00000000                0.08985799
##           path.difference quadratic.path.difference
##               68.55654600                0.61762377
```

Appendix 1

Data Import of original dataset

```
library(ape)
lizards_accession_numbers <- c("JF806202", "HM161150", "FJ356743", "JF806205",
                               "JQ073190", "GU457971", "FJ356741", "JF806207",
                               "JF806210", "AY662592", "AY662591", "FJ356748",
                               "JN112660", "AY662594", "JN112661", "HQ876437",
                               "HQ876434", "AY662590", "FJ356740", "JF806214",
                               "JQ073188", "FJ356749", "JQ073189", "JF806216",
                               "AY662598", "JN112653", "JF806204", "FJ356747",
                               "FJ356744", "HQ876440", "JN112651", "JF806215",
                               "JF806209")
lizards_sequences<-ape::read.GenBank(lizards_accession_numbers)
print(lizards_sequences)
ape::write.dna(lizards_sequences,
               file ="lizard_seqs.fasta",
               format = "fasta",
               append =FALSE,
               nbcol = 6,
               colsep = " ",
               colw = 10)
```

Appendix 1.1

Reading and preparing original data

```
library(seqinr)
# reading original_dataset from fasta file
lizards_sequences = read.fasta("lizard_seqs.fasta")

# preparing data in fasta file (dna sequences include emtpy spaces which will be removed)
for (i in 1:length(lizards_sequences)) {
  lizards_sequences[[i]] = lizards_sequences[[i]][lizards_sequences[[i]] != " "]
}
```

Function code

```
library(seqinr)
get_artificial_sequence_dataset = function(original_dataset) {
  # creating empty varibales which will be filled in following for-loop
  original_base_compositions = list()
  artificial_dataset = list()
  artificial_base_compositions = list()
  a_original = c(); c_original = c(); g_original = c(); t_original = c()
  a_artificial = c(); c_artificial = c(); g_artificial = c(); t_artificial = c()
  for (i in 1:length(original_dataset)) {
    # getting base compositions for each original sequence
    original_base_compositions[[i]] =
      seqinr::count(original_dataset[[i]],1)/length(original_dataset[[i]])
    # creating artificial sequences randomly drawn from the distribution
    # given by the base composition
    artificial_dataset[[as.character(i)]] = sample(x = c("a","c","g","t"),
                                                   size = length(original_dataset[[i]]),
                                                   rep = TRUE,
                                                   prob = original_base_compositions[[i]])
    # creating dataframe to compare base compositions
    # between original and artificial sequences
    artificial_base_compositions[[i]] =
      seqinr::count(artificial_dataset[[i]],1)/length(artificial_dataset[[i]])
    a_original = c(a_original, round(original_base_compositions[[i]][1],2))
    a_artificial = c(a_artificial, round(artificial_base_compositions[[i]][1],2))
    c_original = c(c_original, round(original_base_compositions[[i]][2],2))
    c_artificial = c(c_artificial, round(artificial_base_compositions[[i]][2],2))
    g_original = c(g_original, round(original_base_compositions[[i]][3],2))
    g_artificial = c(g_artificial, round(artificial_base_compositions[[i]][3],2))
    t_original = c(t_original, round(original_base_compositions[[i]][4],2))
    t_artificial = c(t_artificial, round(artificial_base_compositions[[i]][4],2))
  }
  comparison_base_compositions = cbind(
    name_original = names(original_dataset), name_artificial = names(artificial_dataset),
    a_original, a_artificial, c_original, c_artificial,
    g_original, g_artificial, t_original, t_artificial
  )
  rownames(comparison_base_compositions) = 1:nrow(comparison_base_compositions)
  print("comparison of base compositions
        between original and artificial datasets (values rounded): ")
  print(comparison_base_compositions)
  # saving fasta file
  ape::write.dna(artificial_dataset, file ="artificial_dataset_1_1.fasta", format = "fasta",
                 colsep = "")
}
```

Appendix 1.2

Replace the integers by letters

```
for (k in 1:33){
sequences_artificial[[k]][sequences_artificial[[k]] == 1] = "a"
sequences_artificial[[k]][sequences_artificial[[k]] == "2"] = "c"
sequences_artificial[[k]][sequences_artificial[[k]] == "3"] = "g"
sequences_artificial[[k]][sequences_artificial[[k]] == "4"] = "t"
}
```

Appendix 2

Appendix 2.3

```
library(seqinr)
library(DECIPHER)
library(plsgenomics)
library(ape)

# getting all datasets in DNAStringSet format

  # original dataset
    # readAAStringSet-function needs path of fasta file as input. The original
    # dataset needs to be prepared and saved so that the fasta file does not
    # inlcude whitespaces anymore.
      # reading original_dataset from fasta file
      lizards_sequences = read.fasta("lizard_seqs.fasta")
      # preparing data in fasta file (dna sequences include emtpy spaces which will be removed)
      for (i in 1:length(lizards_sequences)) {
        lizards_sequences[[i]] = lizards_sequences[[i]][lizards_sequences[[i]] != " "]
      }
      # saving prepared fasta file
      ape::write.dna(lizards_sequences, file ="lizards_sequences_no_whitespaces.fasta",
                     format = "fasta", colsep = "")
    # reading prepared fasta file as biostrings-object
    lizards_sequences = readDNAStringSet("lizards_sequences_no_whitespaces.fasta")

  # artificial_dataset_1_1
  artificial_dataset_1_1 = readDNAStringSet("artificial_dataset_1_1.fasta")

  # artificial_dataset_1_2
  artificial_dataset_1_2 = readDNAStringSet("artificial_dataset_1_2.fasta")

# alligning sequences for each dataset
sequence_alligning = function(dataset, name) {
  # alligning process
  sequences_alligned = AlignSeqs(dataset)
  # creating distance matrix
  dm_sequences_alligned = DistanceMatrix(sequences_alligned)
  # creating matrix heatmap
  heatmap_dm_sequences_alligned = matrix.heatmap(dm_sequences_alligned)
  dev.copy(png,paste("heatmap_", name, ".png", sep=""))
  dev.off()
  return(sequences_alligned)
}
```

```r
lizards_sequences_alligned = sequence_alligning(dataset = lizards_sequences,
                                               name = "lizards_sequences")
artificial_dataset_1_1_alligned = sequence_alligning(artificial_dataset_1_1,
                                                    name = "artificial_dataset_1_1")
artificial_dataset_1_2_alligned = sequence_alligning(artificial_dataset_1_2,
                                                    name = "artificial_dataset_1_2")
```

Appendix 3

Appendix 3.1

```r
library(seqinr)
library(DECIPHER)
library(plsgenomics)
library(ape)

# creating phylotrees
create_phylotree = function(dataset_name) {
  distanceMatrix = readRDS(paste0("distanceMatrix_", dataset_name, ".RDS"))
  tree = nj(distanceMatrix)
  png(paste("phylotree_", dataset_name, ".png", sep=""))
  plot(tree)
  dev.off()
  return(tree)
}
tree_lizards_sequences = create_phylotree("lizards_sequences")
tree_artificial_dataset_1_1 = create_phylotree("artificial_dataset_1_1")
tree_artificial_dataset_1_2 = create_phylotree("artificial_dataset_1_2")

# performing bootstrap analysis
bootstrap_analysis = function(dataset_name, tree_object) {
  distanceMatrix = readRDS(paste0("distanceMatrix_", dataset_name, ".RDS"))
  bootstrap_result = boot.phylo(phy = tree_object,
                                x = distanceMatrix,
                                FUN = function(x) {
                                  nj(x)
                                })
  png(paste("bootstrap_phylotree_", dataset_name, ".png", sep=""))
  plot(tree_object)
  nodelabels(bootstrap_result, cex=.6)
  dev.off()
}
bootstrap_analysis("lizards_sequences", tree_lizards_sequences)
bootstrap_analysis("artificial_dataset_1_1", tree_artificial_dataset_1_1)
bootstrap_analysis("artificial_dataset_1_2", tree_artificial_dataset_1_2)
```

Appendix 3.2

```r
library(phangorn)
compare_phylotrees = function(phylotree1, phylotree2) {
  if(all(phylotree1$tip.label == phylotree2$tip.label)) {
    comparePhylo(phylotree1, phylotree2)
    treedist(phylotree1, phylotree2)
  } else {
    comparePhylo(phylotree1, phylotree2)
  }
}
# Comparing tree_lizards_sequences & tree_artificial_dataset_1_1
compare_phylotrees(tree_lizards_sequences, tree_artificial_dataset_1_1)
# Comparing tree_lizards_sequences & tree_artificial_dataset_1_2
compare_phylotrees(tree_lizards_sequences, tree_artificial_dataset_1_2)
# Comparing tree_artificial_dataset_1_1 & tree_artificial_dataset_1_2
compare_phylotrees(tree_artificial_dataset_1_1, tree_artificial_dataset_1_2)
```