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. The R code for the reading process can be found in Appendix 1.1 (Reading original data).

After that, 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:

```r
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"     "0.29"       "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"     "0.29"       "0.2"
##   c_artificial g_original g_artificial t_original t_artificial
## 1 "0.21"       "0.24"     "0.23"       "0.26"     "0.27"
## 2 "0.21"       "0.23"     "0.23"       "0.24"     "0.26"
## 3 "0.21"       "0.23"     "0.23"       "0.24"     "0.26"
## 4 "0.2"        "0.24"     "0.25"       "0.26"     "0.27"
## 5 "0.19"       "0.24"     "0.25"       "0.26"     "0.27"
```

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).

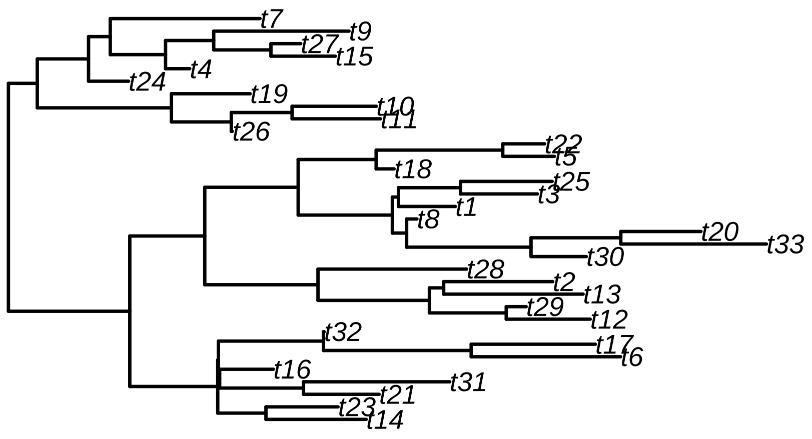
Question 1.2

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.

2. We had to choose the lengths of the sequences. We chose the average length of the origianl sequences and used this length for every artificial sequence.

```
# calculating average length of original sequences
avg_length = c()
for (seq in 1:33) {
  avg_length = c(avg_length, length(lizards_sequences[[seq]]))
}
avg_length = mean(avg_length)
```

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

```
sequences_artificial <- simSeq(tree, l = avg_length, 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

```r
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)
}
```

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

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

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

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

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

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

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

```
##
##     a     c     g     t
## 20567 13341 15116 16411
```

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

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

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

```
##
##         a         c         g         t
## 0.3143119 0.2038817 0.2310079 0.2507985
```

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

```
##
##     a     c     g     t
## 16305 16428 16392 16281
```

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

```
## [1] 65406
```

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

```
##
##         a         c         g         t
## 0.2492891 0.2511696 0.2506192 0.2489221
```

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%.

```r
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)
}
```

```r
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)
}
```

```r
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      CA      TT      CT      TC
##    6960    4554    5664    4867    5096    3909    2277    4296    4947
##      CC      GC      GT      AC      GG (Other)    NA's
##    3207    3076    3172    2727    3237    4728    2685
```

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

```
##    AA    AC    AG    AT    CA    CC    CG    CT    GA    GC    GG    GT    TA    TC    TG
## 6383 4247 4796 5133 4194 2780 3107 3256 4792 3027 3418 3867 5186 3281 3788
##   TT
## 4147
```

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

```
##   AA   AC   AG   AT   CA   CC   CG   CT   GA   GC   GG   GT   TA   TC   TG
## 4040 4097 4116 4043 4054 4126 4180 4060 4097 4129 4074 4085 4110 4068 4012
##   TT
## 4082
```

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)
}
```

```r
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")
# 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){
```

```r
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     20     21     18            59        1     30     33
## 2         2     69     48     61           178        2     33     34
## 3         3     79     48     53           180        3     32     27
## 4         4     23     18     20            61        4     26     30
## 5          5     32     30     26             88        5     26     38
## 6          6     25     25     22             72        6     31     37
## 7          7     83     52     47            182        7     33     42
## 8          8     31     12     27             70        8     38     35
## 9          9     24     24     22             70        9     30     43
## 10        10     88     52     54            194       10     32     32
## 11        11     64     51     44            159       11     21     38
## 12        12     73     56     53            182       12     35     34
## 13        13     43     42     71            156       13     29     29
## 14        14     56     52     53            161       14     38     30
## 15        15     62     54     45            161       15     17     25
## 16        16     23     14     17             54       16     20     25
## 17        17     18     29     19             66       17     24     43
## 18        18     65     45     62            172       18     38     36
## 19        19     70     49     51            170       19     29     24
## 20        20     25     23     28             76       20     34     34
## 21        21     41     26     19             86       21     41     26
## 22        22     73     57     45            175       22     41     28
## 23        23     31     36     20             87       23     29     30
## 24        24     19     24     16             59       24     29     25
## 25        25     74     57     58            189       25     30     29
## 26        26     63     59     48            170       26     29     30
## 27        27     27     20     25             72       27     29     30
## 28        28     72     69     64            205       28     34     26
## 29        29     77     52     54            183       29     24     34
## 30        30     24     19     21             64       30     21     28
## 31        31     71     39     39            149       31     31     32
## 32        32     27     15     23             65       32     32     30
## 33        33     18     17     13             48       33     26     28
##    tga_a2 total_count_3
## 1      30            93
## 2      23            90
## 3      31            90
## 4      39            95
## 5      28            92
## 6      39           107
## 7      22            97
## 8      30           103
## 9      33           106
## 10     33            97
## 11     31            90
## 12     28            97
## 13     36            94
## 14     40           108
## 15     27            69
## 16     23            68
## 17     38           105
## 18     42           116
## 19     31            84
## 20     23            91
## 21     30            97
## 22     39           108
## 23     34            93
## 24     33            87
## 25     28             87
## 26     33             92
## 27     27             86
## 28     31             91
## 29     31             89
## 30     34             83
## 31     29             92
## 32     31             93
## 33     34             88
```

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.

```
reverse_complemented_lizards <- read.fasta("lizards_reverse_complement.fasta")
taa_reverse <- c()
tag_reverse <- c()
tga_reverse <- c()

for (i in 1:33){
  string <- reverse_complemented_lizards[[i]]
  string <- paste(reverse_complemented_lizards[[i]], collapse = "")
  taa_reverse[i] <-str_count(string, pattern = "taa")
  tag_reverse[i] <- str_count(string, pattern = "tag")
  tga_reverse[i] <- str_count(string, pattern = "tga")
}

names_reverse <- names(reverse_complemented_lizards)
df_reverse <- as.data.frame(cbind(names_reverse, taa_reverse, tag_reverse, tga_reverse, total_count_reve
                              taa_reverse + tag_reverse + tga_reverse))
df_reverse
```

```
##    names_reverse taa_reverse tag_reverse tga_reverse total_count_reverse
## 1       JF806202          11           5          21                  37
## 2       HM161150          28          22          55                 105
## 3       FJ356743          29          25          57                 111
## 4       JF806205          12           5          19                  36
## 5       JQ073190          20          11          30                  61
## 6       GU457971          10           7          20                  37
## 7       FJ356741          26          27          59                 112
## 8       JF806207          11           6          19                  36
## 9       JF806210          11           7          20                  38
## 10      AY662592          24          28          60                 112
## 11      AY662591          26          25          55                 106
## 12      FJ356748          29          26          54                 109
## 13      JN112660          27          23          59                 109
## 14      AY662594          27          24          55                 106
## 15      JN112661          31          22          55                 108
## 16      HQ876437           9          10          17                  36
## 17      HQ876434          10           8          20                  38
## 18      AY662590          35          29          50                 114
## 19      FJ356740          30          26          56                 112
## 20      JF806214          11          10          18                  39
## 21      JQ073188          22           8          29                  59
## 22      FJ356749          27          26          64                 117
## 23      JQ073189          21          10          30                   61
## 24      JF806216          10           8          18                   36
## 25      AY662598          29          29          61                  119
## 26      JN112653          17          20          51                   88
## 27      JF806204          10           7          20                   37
## 28      FJ356747          31          31          56                  118
## 29      FJ356744          31          28          61                  120
## 30      HQ876440          11           5          20                   36
## 31      JN112651          25          23          57                  105
## 32      JF806215           8           9          23                   40
## 33      JF806209          10           6          18                   34
```

```
stop_codons <- c(sum(as.numeric(df_all$total_count_2)), sum(as.numeric(df_all$total_count_3)), sum(as.nu
names(stop_codons) <- c("a1", "a2", "reverse_complemented")
stop_codons
```

```
##                   a1                   a2 reverse_complemented
##                  459                  427                  343
```

After reverse complementing the original dataset from the lizards, for this dataset we again determine the number of stopcodons observed. Compared to the two artificially created datasets, the numer of stop codons is smaller. We think this can be explained because creating artificial datasets of sequences completely randomly assigns letters in the sequence, whereas in the reverse complement, the sequence is based on the original sequences. The original sequences will never contain as much stop codons as the artifically created ones.

Question 2.2

```
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 0.3104723 0.2065762 0.2332798 0.2496717
## c 0.3144635 0.2084427 0.2329609 0.2441329
## g 0.3172669 0.2004105 0.2262977 0.2560249
## t 0.3161810 0.2000366 0.2309474 0.2528350
##
##
## $standardError
##             a           c           g           t
## a 0.003886068 0.003169853 0.003368508 0.003484846
## c 0.004855747 0.003953341 0.004179386 0.004278426
## g 0.004583175 0.003642623 0.003870740 0.004117135
## t 0.004390555 0.003492258 0.003752392 0.003926180
##
## $confidenceLevel
## [1] 0.95
##
## $lowerEndpointMatrix
##           a         c         g         t
## a 0.3040803 0.2013623 0.2277391 0.2439396
## c 0.3064765 0.2019400 0.2260865 0.2370955
## g 0.3097283 0.1944189 0.2199309 0.2492528
## t 0.3089591 0.1942923 0.2247753 0.2463770
##
## $upperEndpointMatrix
##           a         c         g         t
## a 0.3168643 0.2117901 0.2388205 0.2554037
## c 0.3224505 0.2149453 0.2398354 0.2511702
## g 0.3248056 0.2064021 0.2326645 0.2627970
## t 0.3234028 0.2057808 0.2371196 0.2592930
```

```
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 0.2479136 0.2514114 0.2525773 0.2480977
## c 0.2468940 0.2512789 0.2545676 0.2472594
## g 0.2500458 0.2519988 0.2486421 0.2493134
## t 0.2525811 0.2500000 0.2465585 0.2508604
##
##
## $standardError
##             a           c           g           t
## a 0.003900405 0.003927824 0.003936921 0.003901853
## c 0.003877651 0.003911933 0.003937449 0.003880519
## g 0.003906488 0.003921715 0.003895508 0.003900763
## t 0.003939853 0.003919670 0.003892598 0.003926409
##
## $confidenceLevel
## [1] 0.95
##
## $lowerEndpointMatrix
##           a         c         g         t
## a 0.2414980 0.2449507 0.2461017 0.2416797
## c 0.2405159 0.2448444 0.2480911 0.2408766
## g 0.2436202 0.2455481 0.2422345 0.2428972
## t 0.2461006 0.2435527 0.2401558 0.2444020
##
## $upperEndpointMatrix
##           a         c         g         t
## a 0.2543292 0.2578721 0.2590530 0.2545157
## c 0.2532722 0.2577135 0.2610441 0.2536423
## g 0.2564714 0.2584494 0.2550496 0.2557296
## t 0.2590616 0.2564473 0.2529613 0.2573187
```

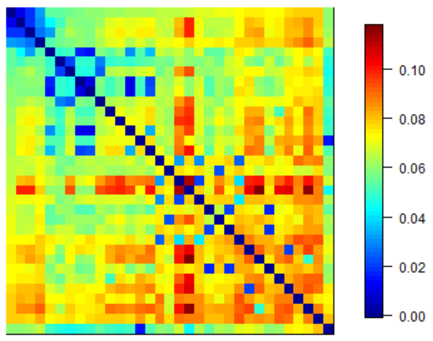
We fitted a first order markov model on all sequences. Our assumption in our simulated datasets is that in the sequence the occurence of a nucleotide does not depend on the rest of the sequence. This violates the limited horizon: which is that the probability of being in a state at time t depends only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

Actually a first order markovchain is not ideal for the artifically created datasets as the letters are randomly assigned. Therefore there is no reason why there should be a connection between two subsequent letters in artifically created sequences.

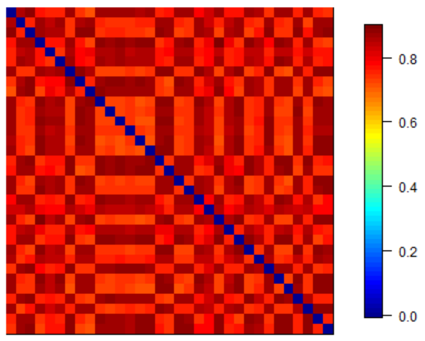
Question 2.3

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) :

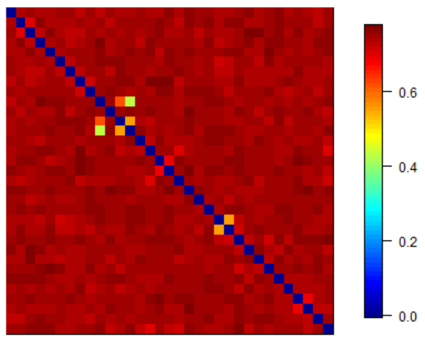
lizards_sequences



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.

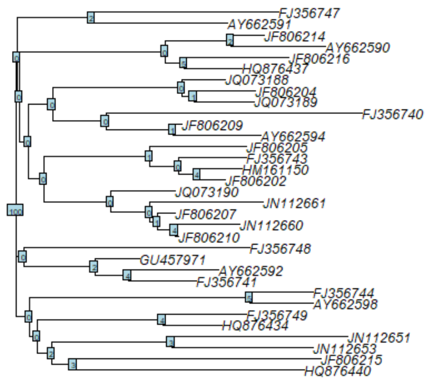
The R code for this Question 2.3 can be found in Appendix 2.3.

Question 3

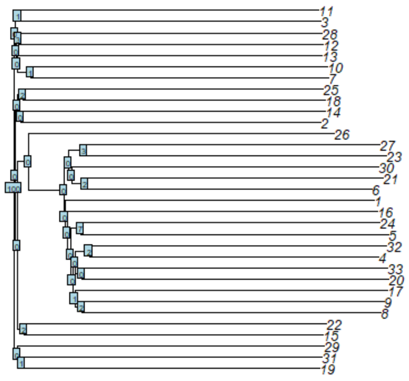
Question 3.1

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.

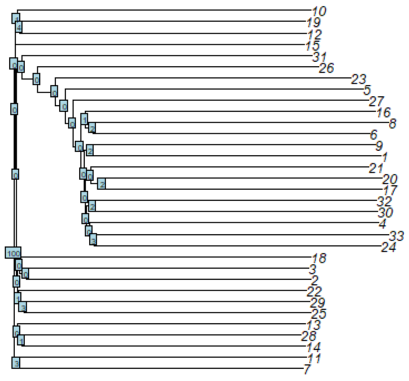
lizards_sequences



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 : t6, t20, t3, t14, t21, t23, t15, t4, t7, t22, t18, t24, t5, t
## 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 : 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18
## Tips in phylotree2 not in phylotree1 : t6, t20, t3, t14, t21, t23, t15, t4, t7, t22, t18, t24, t5, t
## Both trees have the same number of nodes: 31.
## Both trees are unrooted.
## Both trees are not ultrametric.
```

Appendix 1

Data Import of original dataset

```r
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 original data

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

Function code

```r
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
  lizards_sequences = readDNAStringSet("lizard_seqs.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)
  saveRDS(dm_sequences_alligned, paste0("distanceMatrix_", name, ".RDS"))
  # 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)
}

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
artificial_dataset_1_2_alligned = sequence_alligning(artificial_dataset_1_2, name = "artificial_dataset
```

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
```

```r
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)
```