# Bioinformatics - Computer Lab 2

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

*20 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"     "0.3"        "0.2"
## 2 "HM161150"    "2"             "0.31"     "0.3"        "0.21"
## 3 "FJ356743"    "3"             "0.31"     "0.32"       "0.21"
## 4 "JF806205"    "4"             "0.28"     "0.29"       "0.21"
## 5 "JQ073190"    "5"             "0.31"     "0.31"       "0.2"
##   c_artificial g_original g_artificial t_original t_artificial
## 1 "0.18"       "0.24"     "0.25"       "0.26"     "0.27"
## 2 "0.21"       "0.23"     "0.23"       "0.24"     "0.26"
## 3 "0.21"       "0.23"     "0.23"       "0.24"     "0.24"
## 4 "0.21"       "0.24"     "0.22"       "0.26"     "0.28"
## 5 "0.19"       "0.24"     "0.21"       "0.26"     "0.29"
```

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

```
##
## Attaching package: 'ape'
```

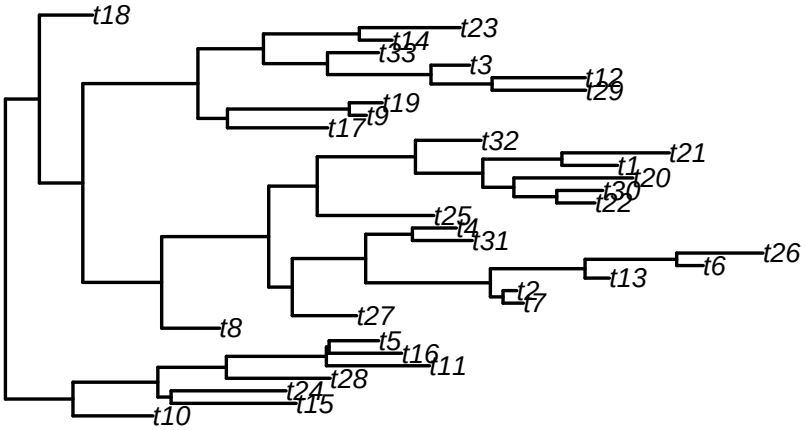
```
## The following objects are masked from 'package:seqinr':
##
##     as.alignment, consensus
```

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)/length(original_dataset[[i]])
}

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)/length(artificial_sequences_1[[i]])
}

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)/length(artificial_sequences_2[[i]])
}
```

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

```
##
##        a        c        g        t
## 9.372287 6.239920 7.068938 7.764488
```

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

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

```
## 10.182870  6.746247  7.664442  8.406441
```

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

```
##
##        a        c        g        t
## 8.279457 8.163184 8.121318 8.436041
```

```
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)
}
```

```
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
```

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

```
##   AA   AC   AG   AT   CA   CC   CG   CT   GA   GC   GG   GT   TA   TC   TG
## 6384 4241 4692 5079 4210 2696 3127 3340 4708 3119 3467 3898 5101 3313 3902
##   TT
## 4125
```

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

```
##   AA   AC   AG   AT   CA   CC   CG   CT   GA   GC   GG   GT   TA   TC   TG
## 4014 4045 4034 4275 4025 3985 4033 4147 4115 4028 3928 4099 4208 4137 4170
##   TT
## 4159
```

```
# 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")
```

```
library(protr)
```

```
##
```

```
## Attaching package: 'protr'
```

```
## The following objects are masked from 'package:rDNAse':
##
##     parGOSim, parSeqSim, readFASTA, twoGOSim, twoSeqSim
```

```
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)
}
```

```
library(protr)
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)
}
```

```
library(protr)
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)
}
```

## Question 2.2

```
library(markovchain)
```

```
## Package:  markovchain
## Version:  0.6.9.12
## Date:     2018-08-23
## BugReport: http://github.com/spedygiorgio/markovchain/issues
```

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

```
## $estimate
## MLE Fit
##  A  9 - 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
##   0.00000000 0.3241607 0.1915245 0.24179050 0.000000e+00 3.669052e-04
## a 0.08640039 0.3050723 0.1580005 0.25263416 4.900760e-05 1.960304e-04
## c 0.07956155 0.3480725 0.2288420 0.04630527 0.000000e+00 3.728283e-04
## g 0.08763673 0.3600928 0.1882002 0.17374876 6.629102e-05 3.314551e-04
## m 0.00000000 0.0000000 0.0000000 0.66666667 0.000000e+00 0.000000e+00
## r 0.00000000 0.4117647 0.1764706 0.11764706 0.000000e+00 0.000000e+00
## s 1.00000000 0.0000000 0.0000000 0.00000000 0.000000e+00 0.000000e+00
## t 0.07883389 0.1400547 0.1957486 0.32572123 6.073489e-05 6.073489e-05
## y 0.00000000 0.3333333 0.2000000 0.13333333 0.000000e+00 0.000000e+00
##              s         t           y
##   0.000000e+00 0.2421574 0.0000000000
## a 0.000000e+00 0.1974026 0.0002450380
## c 0.000000e+00 0.2966967 0.0001491313
## g 0.000000e+00 0.1895260 0.0003977461
## m 0.000000e+00 0.3333333 0.0000000000
## r 0.000000e+00 0.2941176 0.0000000000
## s 0.000000e+00 0.0000000 0.0000000000
## t 6.073489e-05 0.2593380 0.0001214698
## y 0.000000e+00 0.3333333 0.0000000000
##
##
## $standardError
##                         a           c           g           m
##   0.000000000 0.007711557 0.005927534 0.006660112 0.000000e+00
## a 0.002057736 0.003866634 0.002782665 0.003518663 4.900760e-05
## c 0.002435685 0.005094532 0.004130830 0.001858167 0.000000e+00
## g 0.002410296 0.004885787 0.003532136 0.003393815 6.629102e-05
## m 0.000000000 0.000000000 0.000000000 0.471404521 0.000000e+00
## r 0.000000000 0.155632430 0.101885342 0.083189033 0.000000e+00
## s 1.000000000 0.000000000 0.000000000 0.000000000 0.000000e+00
## t 0.002188143 0.002916540 0.003448009 0.004447768 6.073489e-05
## y 0.000000000 0.149071198 0.115470054 0.094280904 0.000000e+00
##              r           s           t           y
##   2.594411e-04 0.000000e+00 0.006665163 0.000000e+00
## a 9.801519e-05 0.000000e+00 0.003110342 1.095843e-04
## c 1.667339e-04 0.000000e+00 0.004703550 1.054518e-04
## g 1.482312e-04 0.000000e+00 0.003544555 1.623792e-04
## m 0.000000e+00 0.000000e+00 0.333333333 0.000000e+00
## r 0.000000e+00 0.000000e+00 0.131533410 0.000000e+00
## s 0.000000e+00 0.000000e+00 0.000000000 0.000000e+00
## t 6.073489e-05 6.073489e-05 0.003968736 8.589211e-05
```

```
## y 0.000000e+00 0.000000e+00 0.149071198 0.000000e+00
##
## $confidenceLevel
## [1] 0.95
##
## $lowerEndpointMatrix
##                       a           c          g m           r s          t
##   0.00000000 0.31147632 0.181774565 0.23083559 0 0.000000e+00 0 0.23119418
## a 0.08301572 0.29871224 0.153423414 0.24684647 0 3.480974e-05 0 0.19228654
## c 0.07555521 0.33969272 0.222047385 0.04324886 0 9.857546e-05 0 0.28896009
## g 0.08367214 0.35205640 0.182390353 0.16816643 0 8.763643e-05 0 0.18369574
## m 0.00000000 0.00000000 0.000000000 0.00000000 0 0.000000e+00 0 0.00000000
## r 0.00000000 0.15577214 0.008884115 0.00000000 0 0.000000e+00 0 0.07776444
## s 0.00000000 0.00000000 0.000000000 0.00000000 0 0.000000e+00 0 0.00000000
## t 0.07523472 0.13525738 0.190077087 0.31840530 0 0.000000e+00 0 0.25281000
## y 0.00000000 0.08813303 0.010068663 0.00000000 0 0.000000e+00 0 0.08813303
##               y
##   0.000000e+00
## 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
##   0.00000000 0.3368451 0.2012744 0.25274541 0.0000000000 0.0007936478
## a 0.08978507 0.3114323 0.1625776 0.25842184 0.0001296179 0.0003572510
## c 0.08356790 0.3564522 0.2356366 0.04936168 0.0000000000 0.0006470811
## g 0.09160131 0.3681292 0.1940100 0.17933109 0.0001753300 0.0005752738
## m 0.00000000 0.0000000 0.0000000 1.00000000 0.0000000000 0.0000000000
## r 0.00000000 0.6677573 0.3440571 0.25448084 0.0000000000 0.0000000000
## s 1.00000000 0.0000000 0.0000000 0.00000000 0.0000000000 0.0000000000
## t 0.08243306 0.1448519 0.2014200 0.33303715 0.0001606349 0.0001606349
## y 0.00000000 0.5785336 0.3899313 0.28841162 0.0000000000 0.0000000000
##               s         t            y
##   0.0000000000 0.2531206 0.0000000000
## a 0.0000000000 0.2025187 0.0004252881
## c 0.0000000000 0.3044334 0.0003225840
## g 0.0000000000 0.1953563 0.0006648361
## m 0.0000000000 0.8816179 0.0000000000
## r 0.0000000000 0.5104709 0.0000000000
## s 0.0000000000 0.0000000 0.0000000000
## t 0.0001606349 0.2658660 0.0002627497
## y 0.0000000000 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 0.3130025 0.2079329 0.2300451 0.2490194
## c 0.3148134 0.2016002 0.2338294 0.2497570
## g 0.3098999 0.2053054 0.2282122 0.2565824
## t 0.3102609 0.2015084 0.2373335 0.2508971
##
##
## $standardError
##             a           c           g           t
## a 0.003917432 0.003192928 0.003358412 0.003494171
## c 0.004851904 0.003882676 0.004181529 0.004321598
## g 0.004516513 0.003676146 0.003875805 0.004109661
## t 0.004344096 0.003500922 0.003799403 0.003906463
##
## $confidenceLevel
## [1] 0.95
##
## $lowerEndpointMatrix
##           a         c         g         t
## a 0.3065589 0.2026810 0.2245210 0.2432720
## c 0.3068328 0.1952138 0.2269514 0.2426486
## g 0.3024709 0.1992587 0.2218371 0.2498226
## t 0.3031155 0.1957499 0.2310840 0.2444716
##
## $upperEndpointMatrix
##           a         c         g         t
## a 0.3194462 0.2131848 0.2355692 0.2547668
## c 0.3227941 0.2079867 0.2407074 0.2568654
## g 0.3173290 0.2113521 0.2345873 0.2633422
## t 0.3174063 0.2072669 0.2435830 0.2573227
```

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

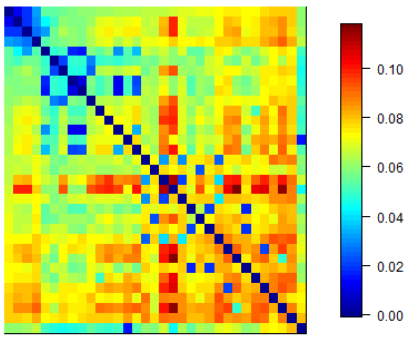
```
## $estimate
## MLE Fit
##  A  4 - dimensional discrete Markov Chain defined by the following states:
##  a, c, g, t
##  The transition matrix  (by rows)  is defined as follows:
##           a         c         g         t
## a 0.2452346 0.2471285 0.2464565 0.2611804
## c 0.2486103 0.2461396 0.2491044 0.2561458
## g 0.2544836 0.2491033 0.2429190 0.2534941
## t 0.2523690 0.2481108 0.2500900 0.2494303
##
##
## $standardError
##             a           c           g           t
## a 0.003870732 0.003885650 0.003880363 0.003994592
## c 0.003918647 0.003899126 0.003922539 0.003977591
## g 0.003967116 0.003924955 0.003875928 0.003959396
## t 0.003890434 0.003857474 0.003872828 0.003867717
##
```

```
## $confidenceLevel
## [1] 0.95
##
## $lowerEndpointMatrix
##           a         c         g         t
## a 0.2388678 0.2407372 0.2400739 0.2546098
## c 0.2421647 0.2397261 0.2426524 0.2496032
## g 0.2479583 0.2426473 0.2365437 0.2469815
## t 0.2459698 0.2417659 0.2437197 0.2430684
##
## $upperEndpointMatrix
##           a         c         g         t
## a 0.2516014 0.2535199 0.2528391 0.2677509
## c 0.2550559 0.2525531 0.2555564 0.2626883
## g 0.2610089 0.2555593 0.2492943 0.2600068
## t 0.2587682 0.2544558 0.2564602 0.2557921
```

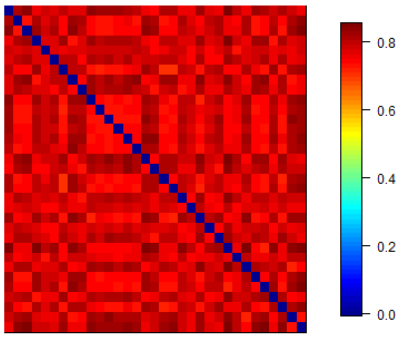
## Question 2.3

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) :

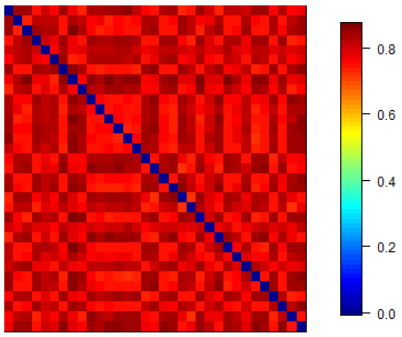
lizards_sequences




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.

---

# 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 2.1

### Appendix 2.2

### 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)
}

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")
```